software AG

# NaturalONE

## Application Testing

Version 8.3.7

March 2016

ADABAS & NATURAL

This document applies to NaturalONE Version 8.3.7.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Copyright © 2009-2016 Software AG, Darmstadt, Germany and/or Software AG USA, Inc., Reston, VA, USA, and/or its subsidiaries and/or its affiliates and/or their licensors.

The name Software AG and all Software AG product names are either trademarks or registered trademarks of Software AG and/or Software AG USA, Inc. and/or its subsidiaries and/or its affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

Detailed information on trademarks and patents owned by Software AG and/or its subsidiaries is located at http://softwareag.com/licenses.

Use of this software is subject to adherence to Software AG's licensing conditions and terms. These terms are part of the product documentation, located at http://softwareag.com/licenses/ and/or in the root installation directory of the licensed product(s).

This software may include portions of third-party products. For third-party copyright notices, license terms, additional rights or restrictions, please refer to "License Texts, Copyright Notices and Disclaimers of Third-Party Products". For certain specific third-party license restrictions, please refer to section E of the Legal Notices available under "License Terms and Conditions for Use of Software AG Products / Copyright and Trademark Notices of Software AG Products". These documents are part of the product documentation, located at http://softwareag.com/licenses and/or in the root installation directory of the licensed product(s).

Use, reproduction, transfer, publication or disclosure is prohibited except as specifically provided for in your License Agreement with Software AG.

**Document ID: ONE-TESTING-DOC-837-20160330**

# Table of Contents


Preface ..... v
1 Release Notes ..... 1
What's New in Version 8.3.1 ..... 2
What's New in Version 8.3.2 ..... 2
What's New in Version 8.3.3 ..... 2
What's New in Version 8.3.4 ..... 2
What's New in Version 8.3.5 ..... 3
2 Prerequisites ..... 5
3 Getting Started ..... 7
General Information ..... 8
Testing a Subprogram Directly ..... 9
Creating a Unit Test ..... 11
Running a Unit Test ..... 14
Generating an Ant Script ..... 15
4 Features of the Test Editors ..... 17
5 Test a Business Service or Subprogram Directly ..... 19
Test a Business Service Directly ..... 20
Test a Subprogram Directly ..... 30
Debug a Subprogram Directly ..... 32
Export/Import Test Data ..... 34
Export Test Data to a CSV File ..... 37
6 Create a Unit Test for a Business Service or Subprogram ..... 43
Enable for Application Testing ..... 44
Create a Unit Test for a Business Service ..... 45
Create a Unit Test for a Subprogram ..... 63
Generate Default Unit Tests ..... 66
Create a New Unit Test Suite ..... 76
Create Summary Reports for Unit Test Log Files ..... 78
Use the Dependencies View ..... 83
7 Create an External Data Unit Test ..... 85
Create the Unit Test ..... 86
Configure Column Mappings and Sample Data ..... 91
8 Create a Sequence Unit Test ..... 95
Create the Unit Test ..... 97
Use the Sequence Unit Test Editor ..... 100
Use the Dependencies View ..... 109
9 Test an External Subroutine ..... 113
Access the Subroutine Tester ..... 114
Test with a Program ..... 115
Test with a Subprogram ..... 116
10 Test a Natural Map ..... 117
11 Setting Preferences for Application Testing ..... 121
Showing the Preferences for Application Testing ..... 122

Set Logging Preferences for Unit Tests ..... 123
Set Server Synchronization Preferences ..... 123
12 Creating Ant Scripts to Run Unit Tests ..... 125
Set Up the Environment ..... 126
Generate the Ant Script and Command Files ..... 126
Define the testsuite Ant Task ..... 130

# Preface

This documentation describes how to test business services, subprograms, subroutines, and maps in the NaturalONE environment. It is organized under the following headings:

| | |
|---|---|
| **Release Notes** | Information on new features and enhancements. |
| **Prerequisites** | Prerequisites for application testing. |
| **Getting Started** | A brief introduction to application testing. How to test a simple subprogram, how to save it as a unit test file, and how to generate an Ant script from the unit test file. |
| **Features of the Test Editors** | Describes the features of the test editors for business services and subprogram, such as navigation options and toolbar icons. |
| **Test a Business Service or Subprogram Directly** | How to run a business service or subprogram by analyzing the parameters in a test editor. |
| **Create a Unit Test for a Business Service or Subprogram** | How to create a Natural unit test for a business service or subprogram. |
| **Create an External Data Unit Test** | How to create a unit test that accepts input and/or validations from a CSV file. |
| **Create a Sequence Unit Test** | How to create a special type of unit test that executes a sequence of test steps in a specified order. |
| **Test an External Subroutine** | How to test an external subroutine using either a subprogram or a program that calls a subprogram. |
| **Test a Natural Map** | How to test a map as you would on the server. |
| **Setting Preferences for Application Testing** | Describes the preferences you can set for the test functions, such as setting preferences for logging unit test results and synchronizing local resources with those on the server. |
| **Creating Ant Scripts to Run Unit Tests** | How to create XML-based Ant scripts to run unit test files. |

# 1 Release Notes


- What's New in Version 8.3.1 ..... 2
- What's New in Version 8.3.2 ..... 2
- What's New in Version 8.3.3 ..... 2
- What's New in Version 8.3.4 ..... 2
- What's New in Version 8.3.5 ..... 3

These *Release Notes* pertain to the Application Testing component of NaturalONE version 8.3.

## What's New in Version 8.3.1

### Debug Subprograms via RPC

You can now use the NaturalONE debug attach server to debug a subprogram in the **Test Subprogram** editor. For information, see *Debug a Subprogram Directly*.

## What's New in Version 8.3.2

### New Connection Parameters in the testsuite Ant Task to Define RPC Credentials

New connection parameters have been added to the testsuite Ant task to define an RPC user ID and password for secured Natural environments. For information, see *Parameters Specified as Nested Elements*.

## What's New in Version 8.3.3

### New Parameters in the testsuite Ant Task to Handle Unit Test Failures

New parameters have been added to the testsuite Ant task that will help control the behavior of the Ant script upon unit test failures. For information, see *Parameters*.

## What's New in Version 8.3.4

### New Connection Parameter to Define that only the RPC Server is to be Used

The new option **Set project steplibs** has been added to the tab which defines the environment in which the test is to be run. It is used to indicate whether the steplibs from the Natural project are set in the RPC server environment. If checked (default), the Natural Development Server is used. If not checked, only the RPC server environment is used, without Natural Development Server. See *Define Connections*.

In addition, the new parameter setProjectSteplibs has been added to the testsuite Ant Task. See *Parameters*.

In order to be compatible with previous versions, the Natural Development Server is used by default.

## What's New in Version 8.3.5

### New Style Sheet

A new style sheet, *natural_mapeditor.css*, is available. If you would like to see the same colors in the output window as in the map editor, you can use this new style sheet instead of the default style sheet *natural.css*. See also ***Test a Natural Map***.

# 2 Prerequisites

When the Application Testing component of NaturalONE has been installed, you can use the test functions supplied with NaturalONE. If this component has not yet been installed, use the Software AG Installer to install it.

The tests are run using the EntireX RPC mechanism. While many details are hidden, you must have some knowledge of EntireX RPC to run the tests.

To test subprograms and business services directly, and to create unit tests for subprograms and business services, a Natural RPC server is required. The Natural Development Server cannot be used in this context. If you are testing items in a project connected to the local Natural runtime environment, a special connection via RPC must be made.

As a business service cannot be tested in the local Natural runtime environment without a full local installation of Natural Business Services, the tests are simulated locally by calling the subprogram directly.

# 3 Getting Started

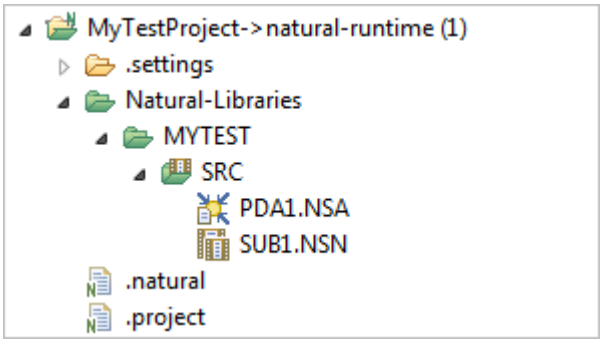

- General Information ..... 8
- Testing a Subprogram Directly ..... 9
- Creating a Unit Test ..... 11
- Running a Unit Test ..... 14
- Generating an Ant Script ..... 15

## General Information

This is a brief introduction to working with the Application Testing component of NaturalONE. It explains how to test a simple subprogram, how to save it as a unit test file, and how to generate an Ant script from the unit test file. You can then use the Ant script, for example, with your automated nightly tests. It is assumed that you use the local RPC server which is automatically started when you start NaturalONE.

**Note:** Testing business services is not in the scope of this introduction. However, this works similarly.

The topics below assume that you have created a Natural project which uses the local Natural runtime. This Natural project contains a library with two objects, a subprogram and a parameter data area (PDA).




Testing is illustrated using the following simple subprogram:

```
DEFINE DATA PARAMETER
USING PDA1
END-DEFINE
#result := #var1 + #var2
END
```

where the subprogram calls the following PDA:

```
DEFINE DATA PARAMETER
 1 #var1 (I4)
 1 #var2 (I4)
 1 #result (I4)
END-DEFINE
```

Do not forget to build your Natural project before you start testing.

## Testing a Subprogram Directly

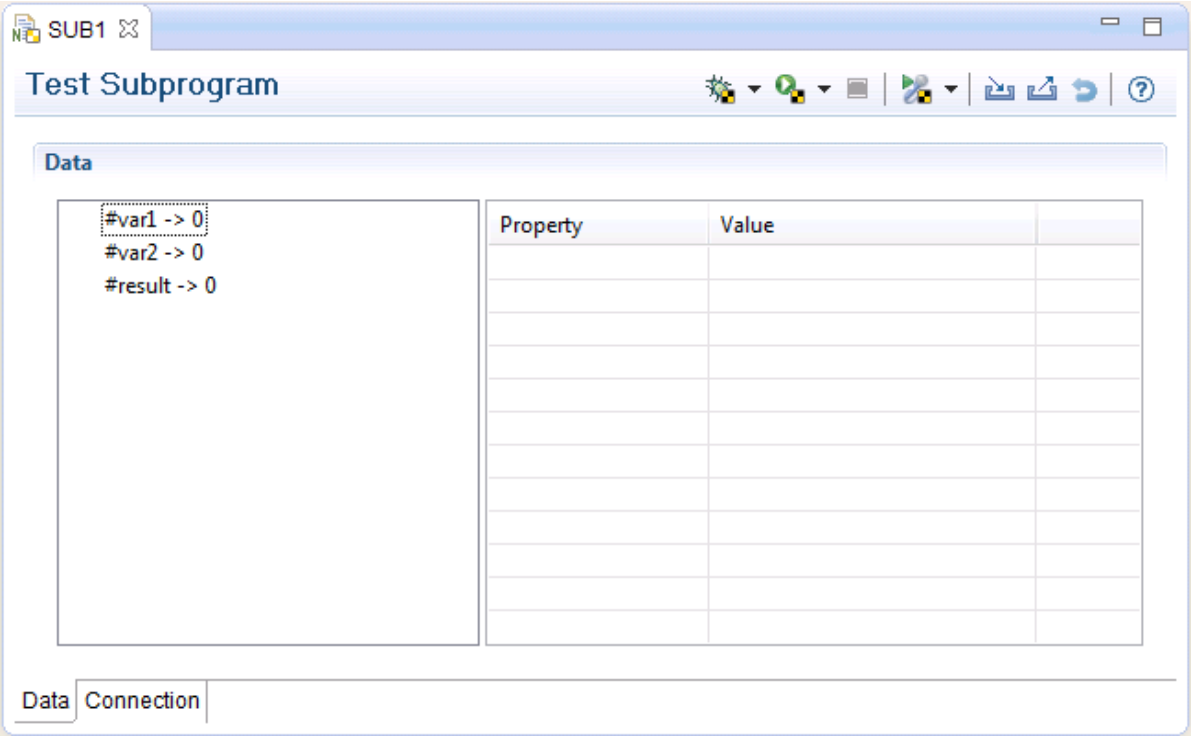
When you test a subprogram directly, you can analyze the parameters in a test editor. You can change the input values, run the test, and verify the return values.

### To test a subprogram

1 In the **Navigator** view, select the subprogram.

2 Invoke the context menu and choose **Testing > Test Subprogram**.

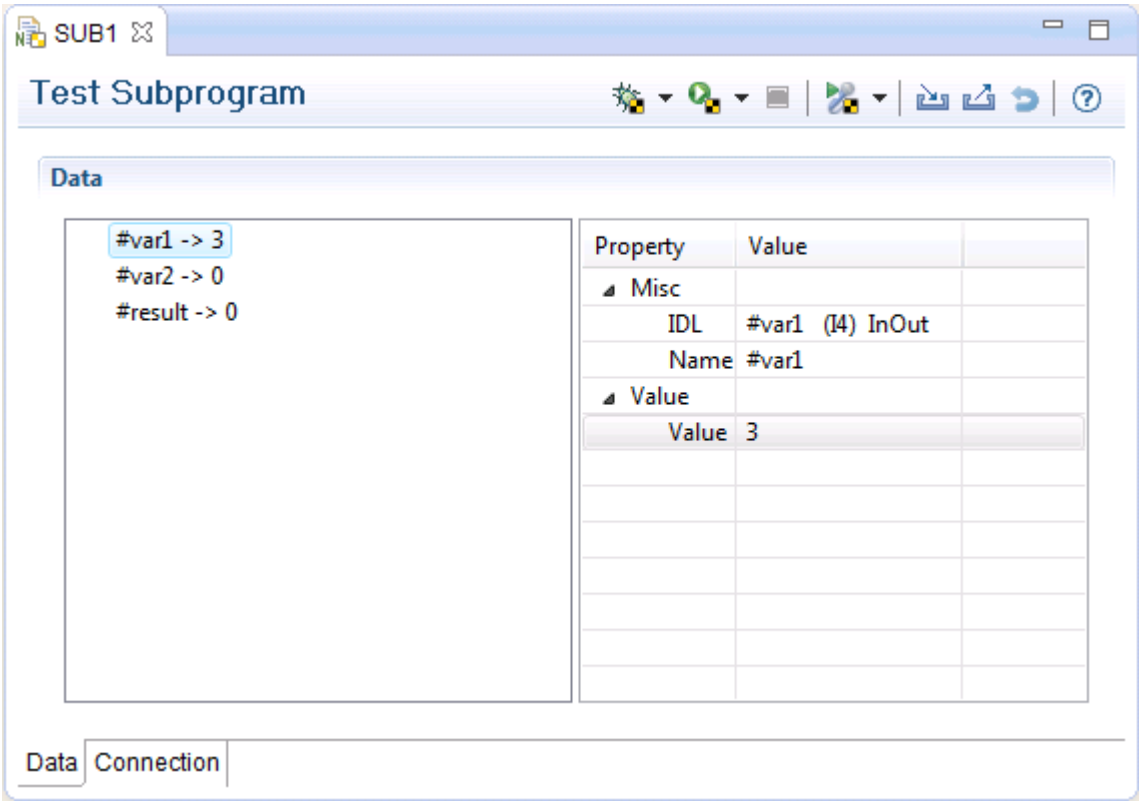
The test editor appears.




3 Select the entry for `#var1` on the left side of the test editor.

Properties are now shown on the right side of the test editor.

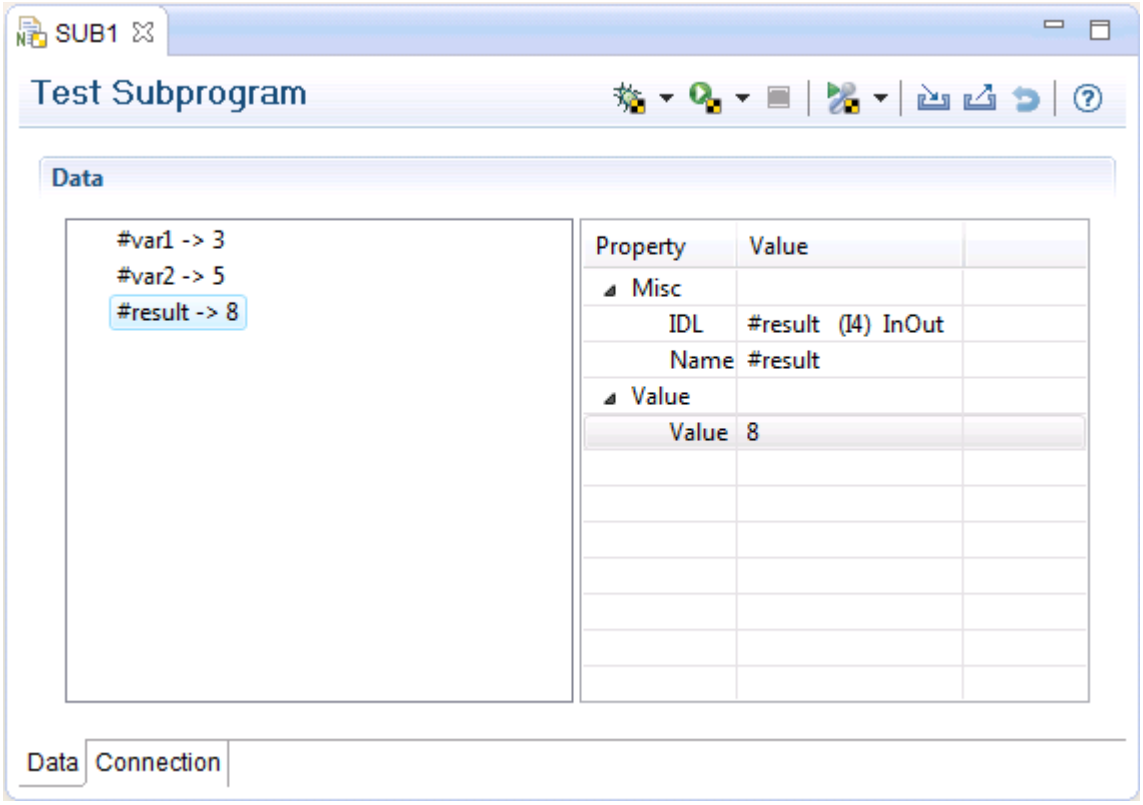
4 Define a value for `#var1` on the right side. Example:




5 Select the entry for `#var2` on the left side of the test editor, and define a value on the right side.

6 Choose the (Run Test) button in the local toolbar of the test editor.

The result value is now shown. Example:




## Creating a Unit Test

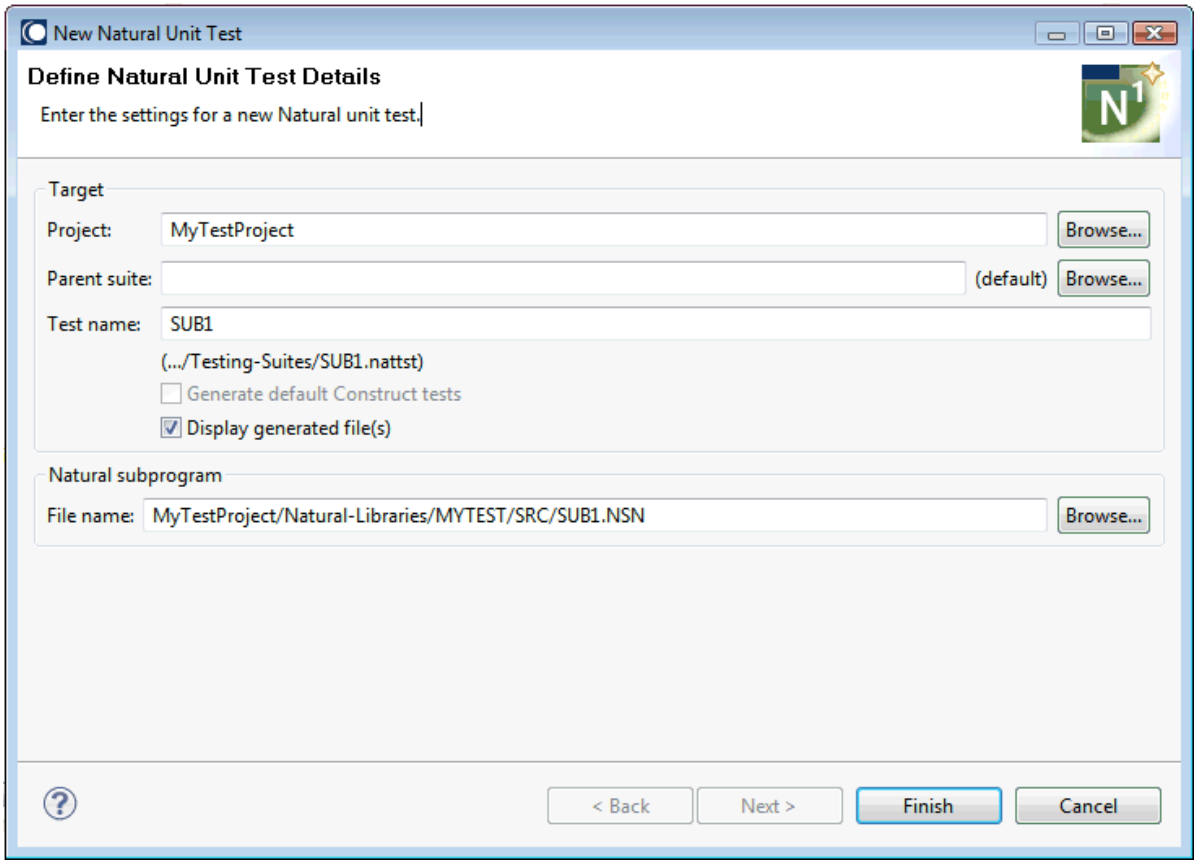
After defining the input and output parameters for the test, you can save this as a unit test.

### To create a unit test

1 Make sure that the editor with your previous test is active.

2 From the **File** menu, choose **Save As**.

The **New Natural Unit Test** dialog appears.

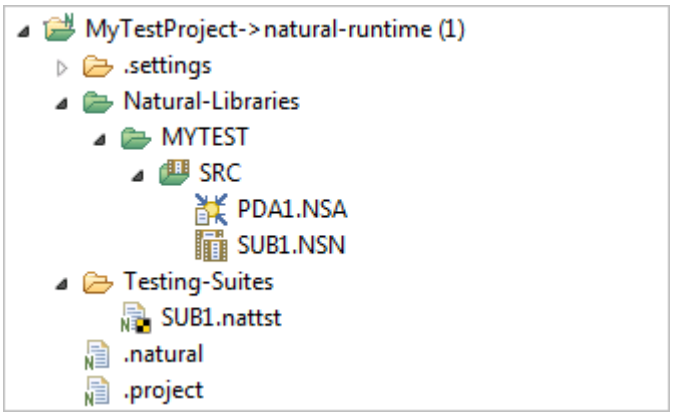



The name of the subprogram is automatically provided as the name for the unit test. For now, you need not change any information.

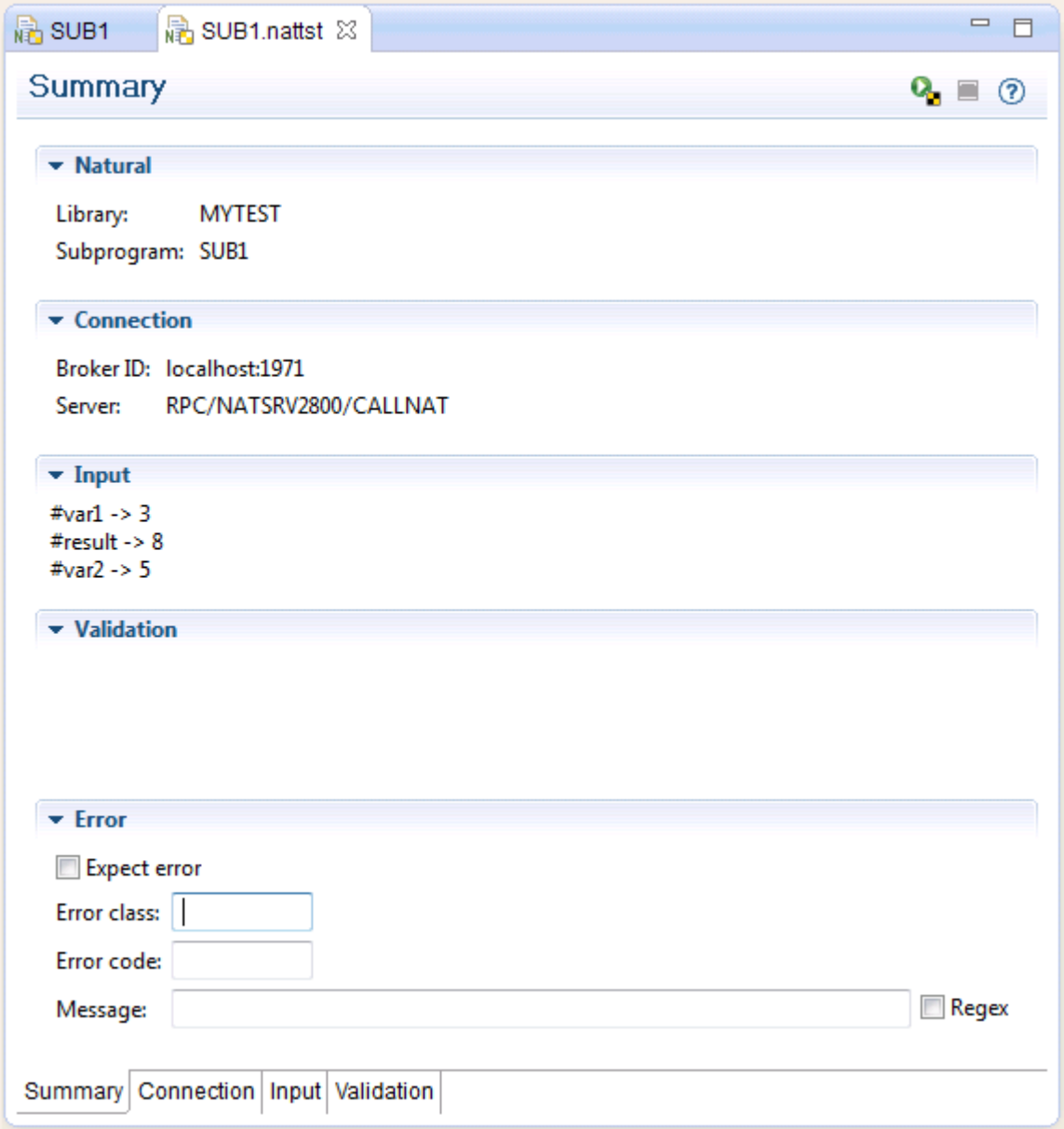
3 Choose the **Finish** button.

A folder named **Testing-Suites** is automatically created in the project. This folder is always created when you create the first unit test in a project.

The unit test is stored in the new **Testing-Suites** folder. Unit tests for subprograms (also called "Natural unit tests)" have the extension *.nattst*.



The generated unit test file is automatically shown in the unit test editor (provided that you have not deselected the **Display generated file(s)** in the **New Natural Unit Test** dialog).




The **Summary** tab shows the information that will be used for the test.

You can save the test for later reuse. For example, you can use it as the basis for an Ant script. Before you save the test, however, you have the possibility the change information on the following tabs:

- **Summary**
  You can allow a test to pass with an expected error.
- **Connection**
  You can define a different RPC environment for your test.
- **Input**
  You can change the input fields that are to be sent to the server.
- **Validation**
  You can configure the fields that are to be tested after the call to the server has been made.

Detailed information these tabs is provided later in this documentation.

4 Use the standard Eclipse functionality to save your changes. For example, press CTRL+S.

## Running a Unit Test

After you have created the unit test, you can run it in order to check whether it works as expected.

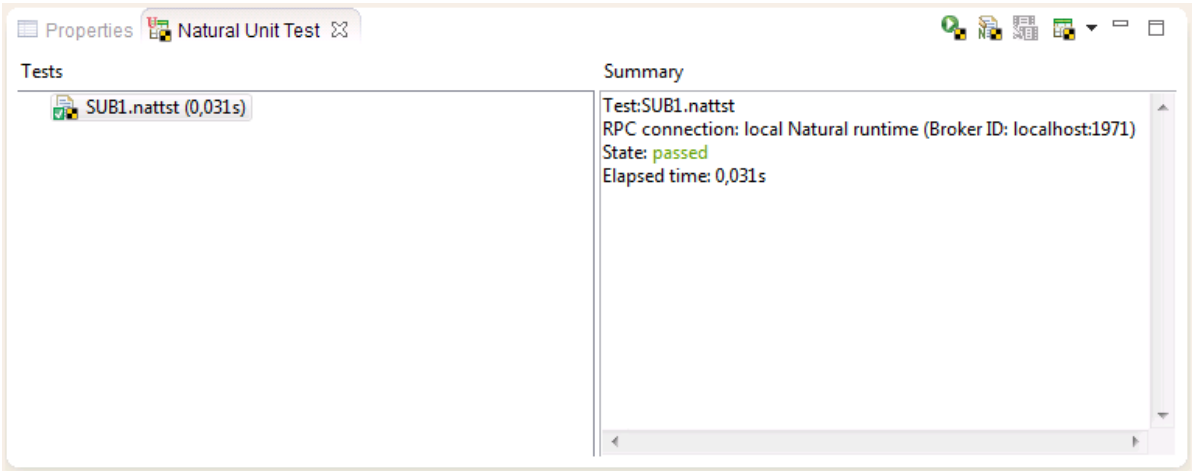
### To run a unit test

- Choose the button in the local toolbar of the unit test editor.

  Or:

  In the **Navigator** view, select the file with the extension *.nattst*, invoke the context menu and choose **Testing > Run Unit Test(s)**.

  The **Natural Unit Test** view is automatically opened the first time you run a unit test. Example:




  When the test was successful, the state "passed" is shown.

## Generating an Ant Script

You can generate XML-based Ant scripts which run your unit test files.

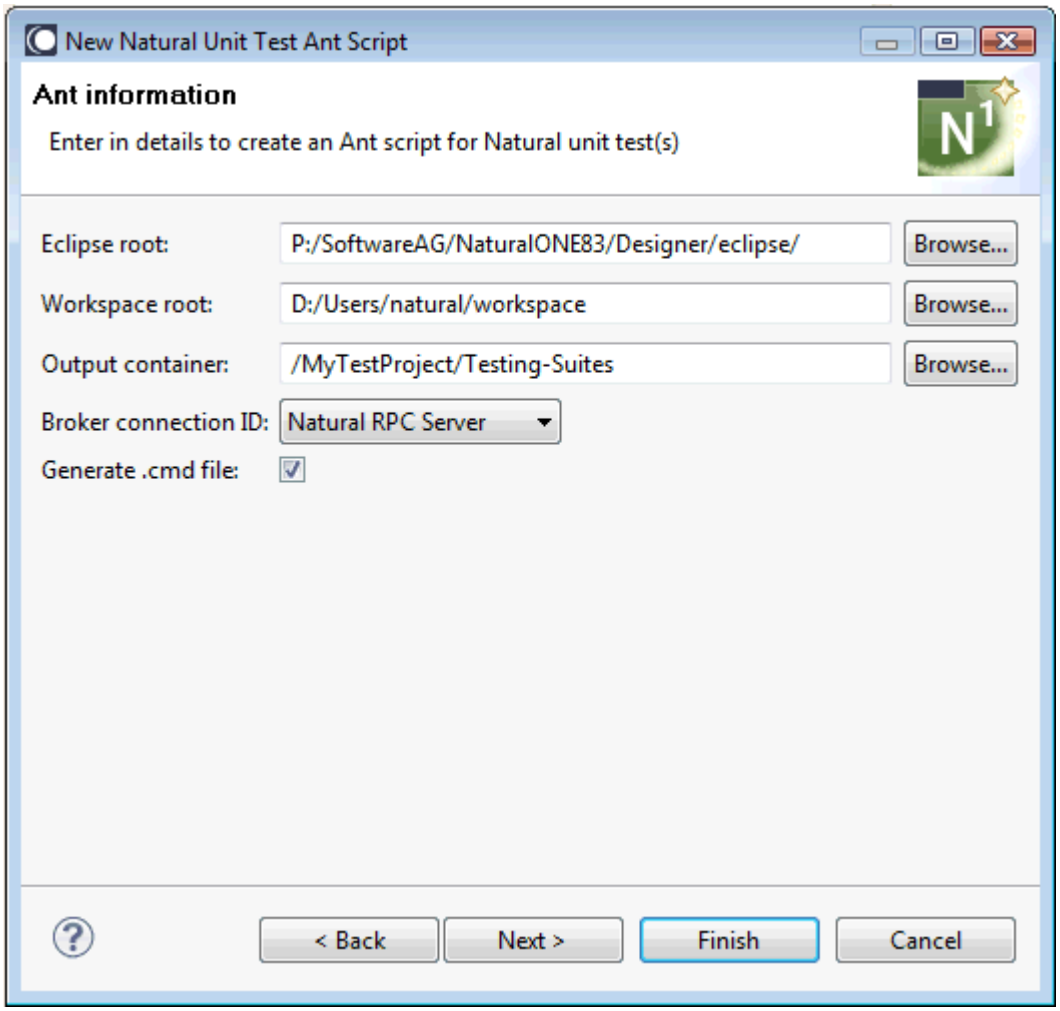
### To generate an Ant script

1 In the **Navigator** view, select the unit test file you have previously created (that is, the file with the extension *.nattst*).

2 Invoke the context menu and choose **New > Other**.

3 In the resulting dialog box, expand **Software AG > Testing** and then choose **Natural Unit Test Ant Script**.

The following dialog appears.




For now, leave the provided settings as they are.

4 Choose the **Finish** button.

The following files are created in the **Testing-Suites** folder:

- *build.xml*

  This file contains the Ant script. It is automatically opened in the Ant editor, and you can now refine the Ant parameters as desired. Detailed information on these parameters is provided later in this documentation.

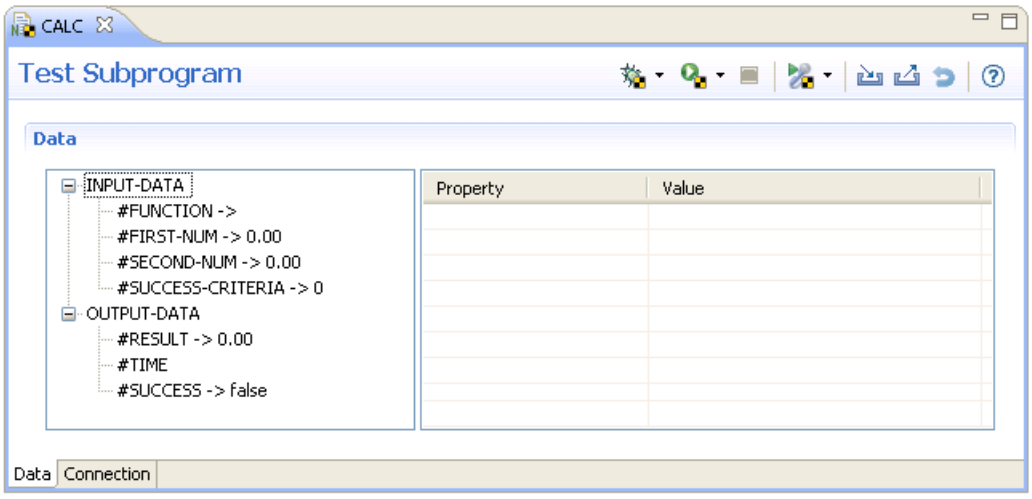
- *run.cmd*

  This file contains the DOS commands for running the script. You run this file outside of Eclipse via the command line.

# 4 Features of the Test Editors

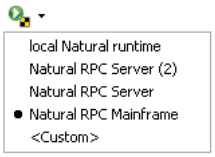
This section describes the features of the test editors, such as navigation options and toolbar icons. The following example shows the **Test Subprogram** editor. The test editors are similar for both business services and subprograms; the main differences between the editors are that the Debug option is not available in the **Test Business Service** editor and you can select the method to test (which can change which subprogram is tested internally).

In this example, the INPUT-DATA and OUTPUT-DATA fields have been expanded:




Keyboard navigation is supported in all editors. In the example above, you can use keys on the keyboard to move from one field to another in the tree view and/or navigate to the table on the right to add or edit values.

The following table describes each of the options available on the editor toolbar:

| Toolbar Icon | Description |
|---|---|
| | Debugs the current subprogram using the NaturalONE debug attach server and the current values defined in the editor. For information, see *Debug a Subprogram Directly*.<br><br>**Note:** This toolbar icon is only visible in the **Test Subprogram** editor and when the **Use debug attach server** option is selected in the Eclipse **Preferences > Software AG > Natural > Debug Attach Settings** window; it is not available in the **Test Business Service** editor. |
| | Runs the current file using the values defined in the editor. Use the down arrow to display the available environments in which to run the test and select a different environment. For example:<br>   |
| | Stops the current test. |
| | Records the test data for export to a CSV file (file extension *.csv*), which can then be used as input for an external data unit test. After selecting this option, either the record function for the test will begin or the **Define External Test Details** panel will be displayed to define the external data unit test. To change details about the recording, select the down arrow. For example: <br><br>The **Define External Test Details** panel is displayed. For more information, see *Export Test Data to a CSV File*. |
| | Exports test data (field names and values) from the data tree in the test editor view to a new or existing test data file (extension *.tstdata*) in the workspace. For information, see *Export Test Data*. |
| | Imports an existing test data file in the workspace to the data tree in the test editor view by matching field names in the imported test data file to field names in the editor tree. For information, see *Import Test Data*. |
| | Resets all data values and structures to their default values. |

# 5 Test a Business Service or Subprogram Directly


- Test a Business Service Directly ..... 20
- Test a Subprogram Directly ..... 30
- Debug a Subprogram Directly ..... 32
- Export/Import Test Data ..... 34
- Export Test Data to a CSV File ..... 37

This section describes how to test a business service or subprogram directly. It provides an easy way to run a business service or subprogram by analyzing the parameters, displaying them in a test editor (tester), and allowing you to change the input values. You can then run the test and verify the return values.

## Test a Business Service Directly

This section describes how to test a business service directly. The following topics are covered:

- Test the Service
- Define Date and Time Details
- Define Connections
- Define Additional RPC Environments
- Save as a Business Service Unit Test

**Note:** The subprograms used for the service must be available locally. If they are not available locally, download them from the server.

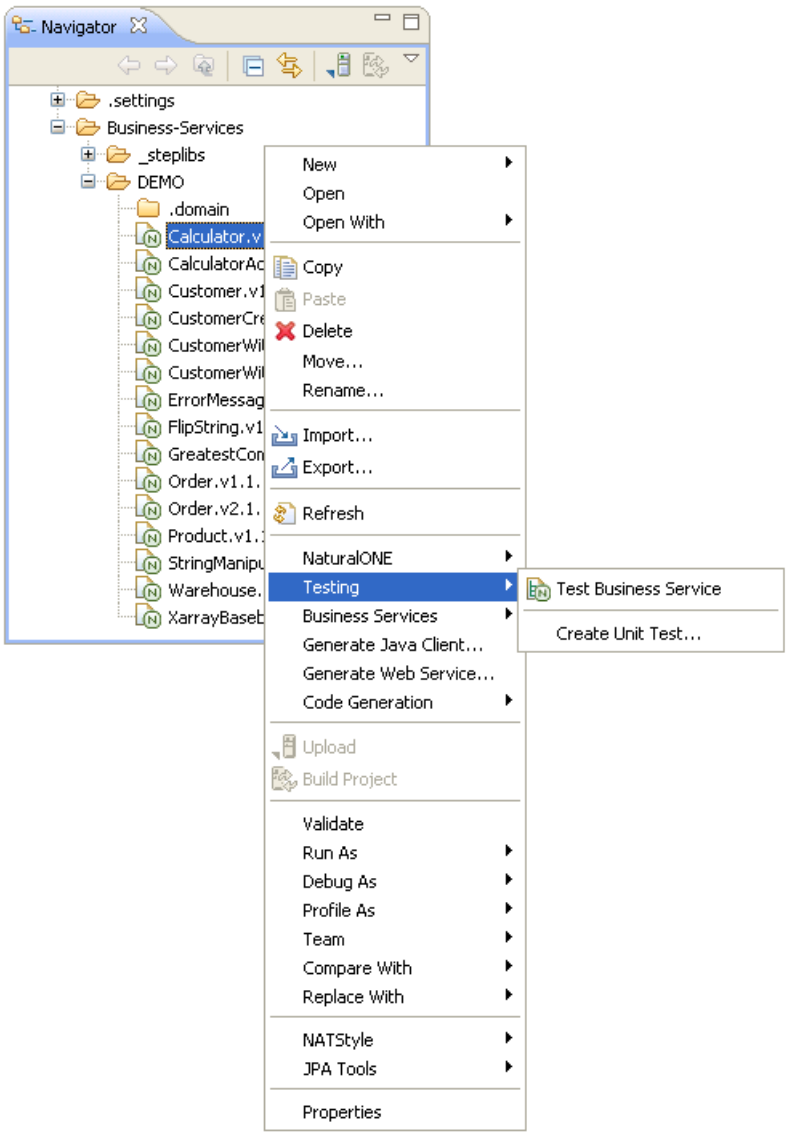
### Test the Service

**To test a business service directly**

1 Open the context menu for the business service in the **Navigator** view.

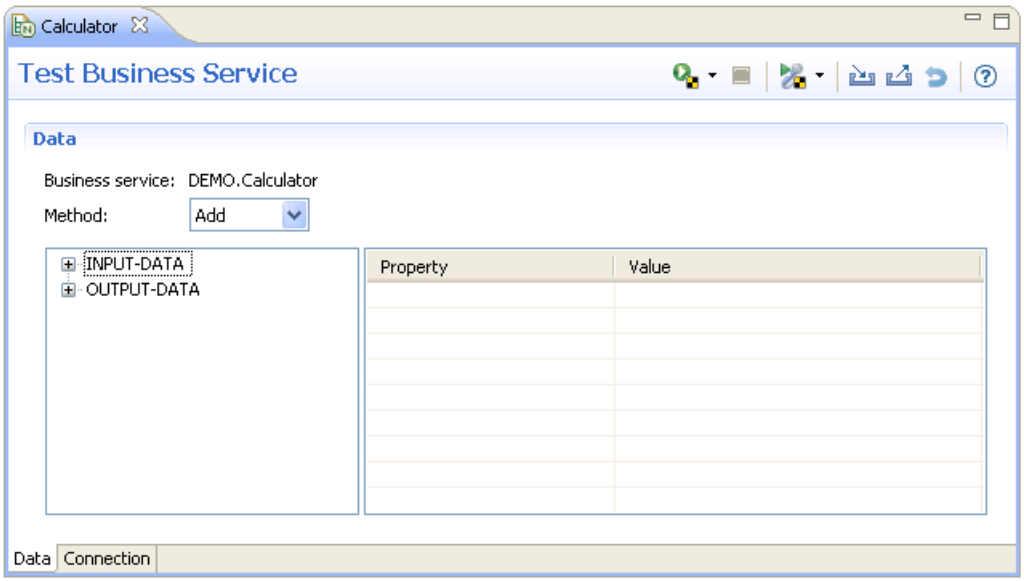
2 Select **Testing**.

The testing options for business services are displayed. For example:




3 Select **Test Business Service**.

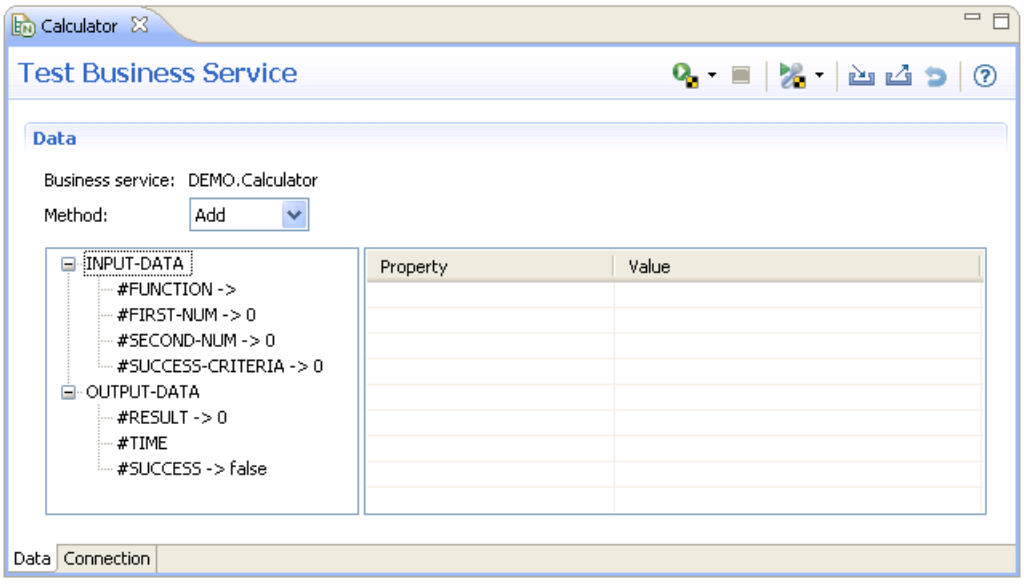
The business service is displayed in the editor view. For example:




**Note:** For information on using this editor, see *Features of the Test Editors*.

4 Expand the **INPUT-DATA** and **OUTPUT-DATA** nodes.

The **Data** tab displays the properties and values defined for each parameter used in the test. For example:




5 Select each input and output field to use in the test and define the value for the `Value` property.

For example:

| Parameter | Value |
|---|---|
| FIRST-NUM | 2 |
| SECOND-NUM | 3 |
| RESULT | 5 |
| SUCCESS | true (select **Value** to change the value from false to true) |

Optionally, you can:

| Task | Procedure |
|---|---|
| Define test data for another method used by the business service. | Select the method in **Method**.<br><br>**Note:** Changing the method may change which subprogram is tested; the parameters may also change. |
| Define input parameters for the test. | Expand the **INPUT-DATA** node and provide input values for each property in **Property** and **Value**. |
| Define output parameters for the test. | Expand the **OUTPUT-DATA** node and provide output values for each property in **Property** and **Value**. |
| Reset all data values and structures to their default values. | Select the **Reset Data** toolbar icon. For example: |
| Enter date and/or time details. | See *Define Date and Time Details*. |
| Run the test in another environment. | See *Define Connections*. |
| Interrupt a test that continues to run with no response. | Select the **Stop Test** toolbar icon. For example: |
| Export and import test data for business services and subprograms. | See *Export/Import Test Data*. |
| Record test data and then export it to a CSV file (file extension *.csv*). | See *Export Test Data to a CSV File*. |

6 Select to start the test.

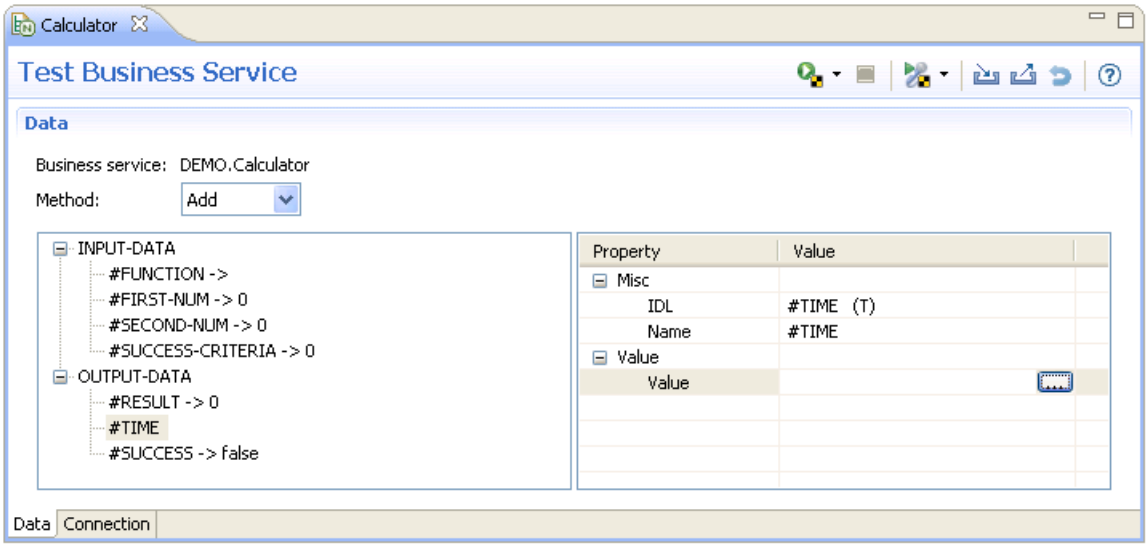
## Define Date and Time Details

When defining the value for a date and/or time field in a subprogram used by a business service, a window is displayed to enter details about the date or time. This section describes how to access and define details about a date or time field.

### To define details about a date or time field

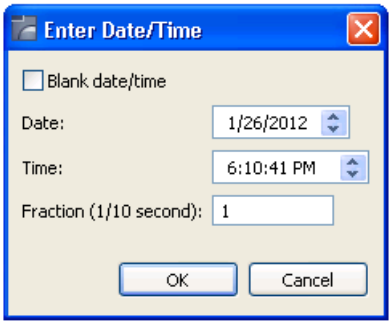
1 Select **Value** for a date or time field in the testing editor.

For example:




2 Select … in the **Value** column.

The **Enter Date/Time** window is displayed. For example:




By default, the current date and time are displayed. Optionally, you can:

| Task | Procedure |
|---|---|
| Blank out date and time information when testing business services or subprograms. | Select **Blank date/time**. |
| Change the date used for the test. | To change the month, select the up or down arrow for **Date**.<br><br>To change the day, select the day portion of **Date** and then select the up or down arrow.<br><br>To change the year, select the year portion of **Date** and then select the up or down arrow. |

| Task | Procedure |
|---|---|
| Change the time used for the test. | To change the hour, select the up or down arrow for **Time**.<br><br>To change the minute, select the minute portion of **Time** and then select the up or down arrow.<br><br>To change the second, select the second portion of **Time** and then select the up or down arrow. |
| Use tenths of a second to define the time used for the test. | Enter the number of tenths of a second in **Fraction**. |

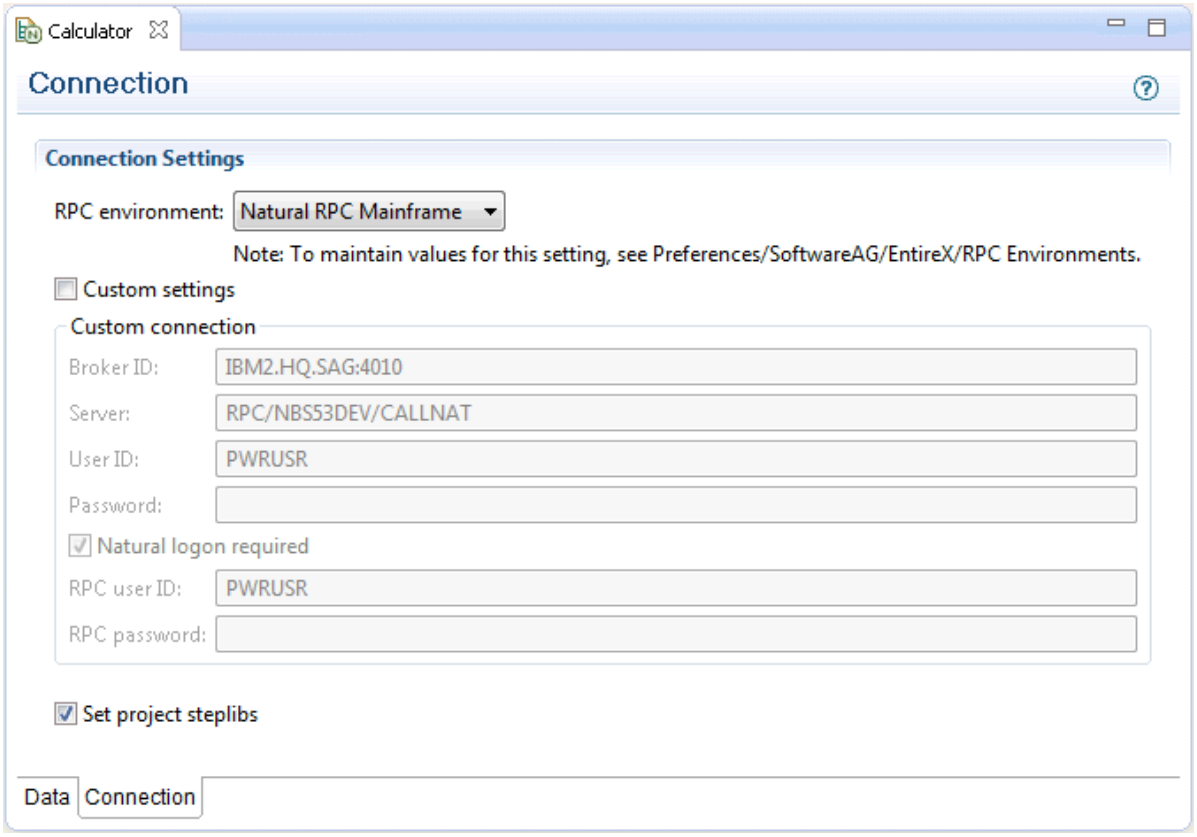
## Define Connections

This section describes the **Connection** tab in the editor view. This tab is used to maintain information about the environment in which the test will run.

### To define the connection settings

1 Select the **Connection** tab for the test.

For example:



This tab shows the current connection settings for the RPC environment. For this example, the settings define a Natural RPC Mainframe environment derived directly from NaturalONE, as well as settings indicating how the RPC server will be started.

2 Select the environment in which to run the test in **RPC environment**.

This value defines the name of an EntireX RPC connection configured in Eclipse.

**Note:** The list of environments is defined in the **Preferences** window for RPC environments. For information on adding additional environments to the list, see *Define Additional RPC Environments*.

Or:

Select **Custom settings** and define the custom connection as follows:

| Setting | Description |
|---|---|
| Broker ID | Broker identifier. Each installation of EntireX is assigned a Broker ID. This number uniquely identifies EntireX to your network. If you do not know the Broker ID, ask the network administrator for your organization. |
| Server | Name of the Broker server used to logically describe a server (rather than the name of the program that implements the server). This allows you to change the program name without affecting the client programs that use the service. |
| User ID | User identifier the server will use to assign the corresponding fields in the EntireX control block when making calls using the EntireX ACI (Advanced Communication Interface). |
| Password | Password value the server will use to assign the corresponding fields in the EntireX control block. |
| Natural logon required | Determines whether a Natural logon is required. |
| RPC user ID | User identifier the EntireX RPC server will use to connect with the Natural server. |
| RPC password | Password value the EntireX RPC server will use to connect with the Natural server. |

3 Use the **Set project steplibs** check box to indicate whether the steplibs from the Natural project are set in the RPC server environment. If checked, the Natural Development Server is used. If not checked, only the RPC server environment is used, without Natural Development Server.

**Note:** Keep in mind that the Natural Development Server used by the project must always be accessible when the value is checked. The Natural Development Server is accessed to check the development mode settings for the steplib consolidation. These steplibs are then passed to the RPC server.

4 Save the connection settings.

## Define Additional RPC Environments

You can define additional RPC environments. Once new environments have been added, they can be selected in **RPC environment** on the **Connection** tab.

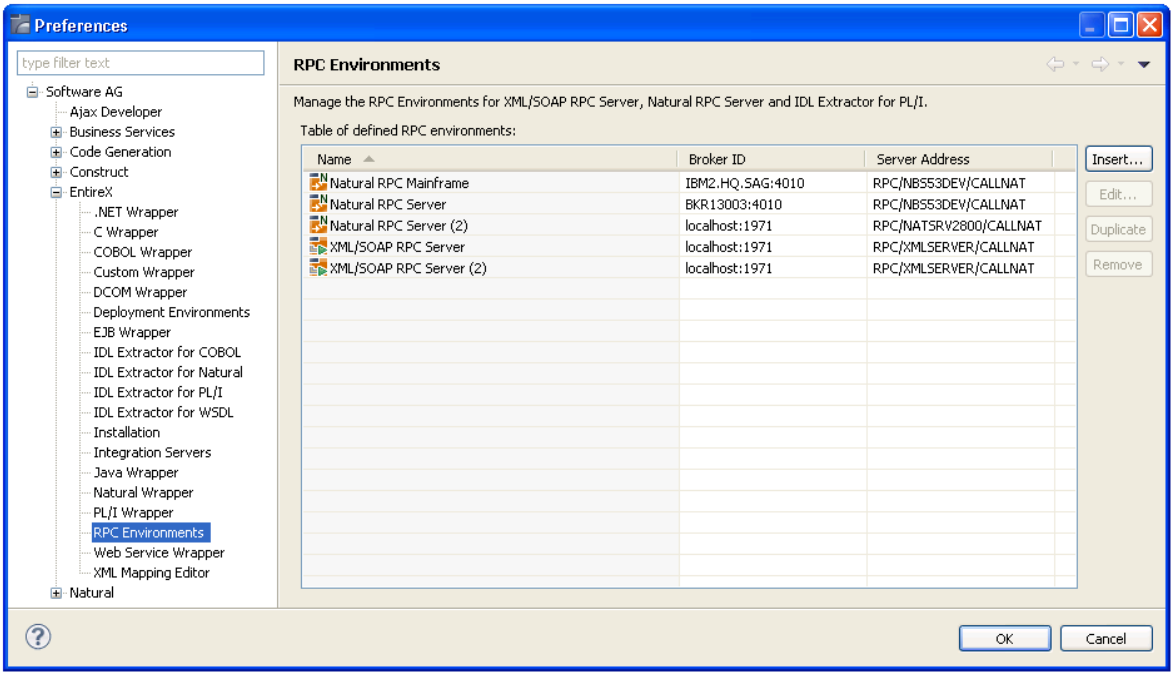
### To define additional RPC environments

1 Select **Preferences** on the **Window** menu. The **Preferences** window is displayed.

2 Expand the **Software AG** node.

3 Select **EntireX > RPC Environments**.

The **RPC Environments** settings are displayed. For example:

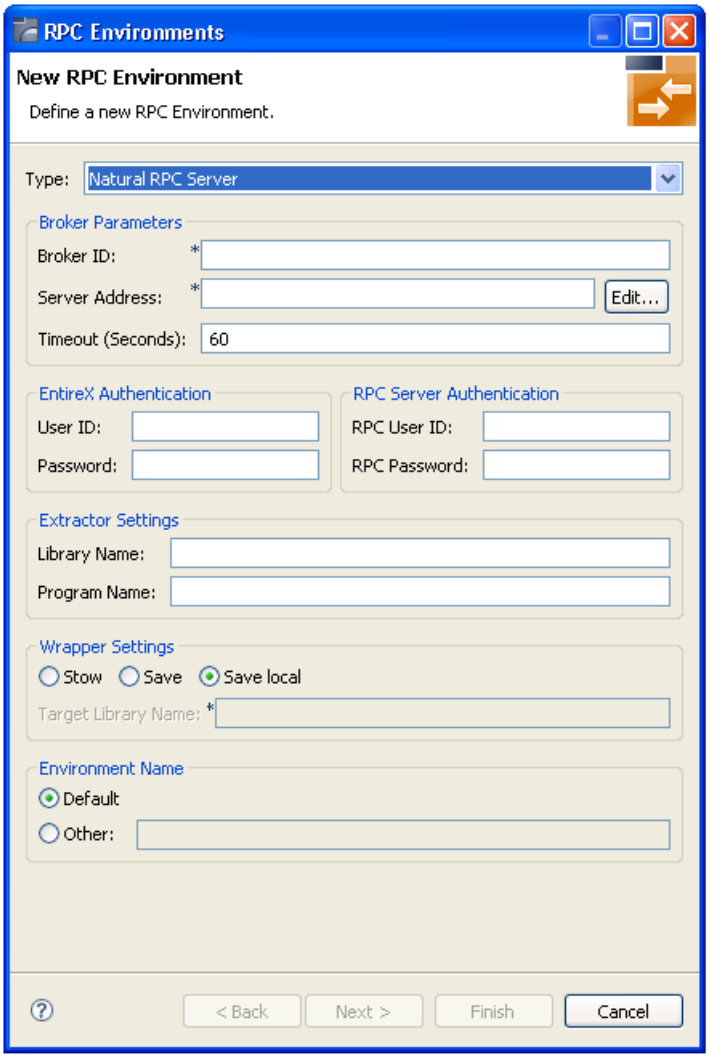



4 Select **Insert**.

The **New RPC Environment** panel is displayed.

5 Select **Natural RPC Server** in **Type**.

The specification fields for this type of server are displayed. For example:




6 Provide the following details about the new environment:

| Section | Description |
|---|---|
| Broker parameters | Type the broker ID, server address, and default timeout values in the fields provided. |
| EntireX authentication | Type the user ID and password for EntireX in the fields provided. |
| RPC server authentication | Type the user ID and password for the RPC server in the fields provided. |
| Extractor settings | Type the name of the library and program from which to extract data in the fields provided. |
| Wrapper settings | If the new environment is not a local environment, select **Stow** or **Save** and provide the name of the library in which to stow or save wrapper subprograms in **Target library name**. |

| Section | Description |
|---|---|
| Environment name | After entering the Broker parameters, the default name of the new environment is displayed in this section. If you do not want to use the default name, select **Other** and provide a new name. |

For more information about the settings on this panel, refer to the EntireX documentation.

7 Select **Finish**.

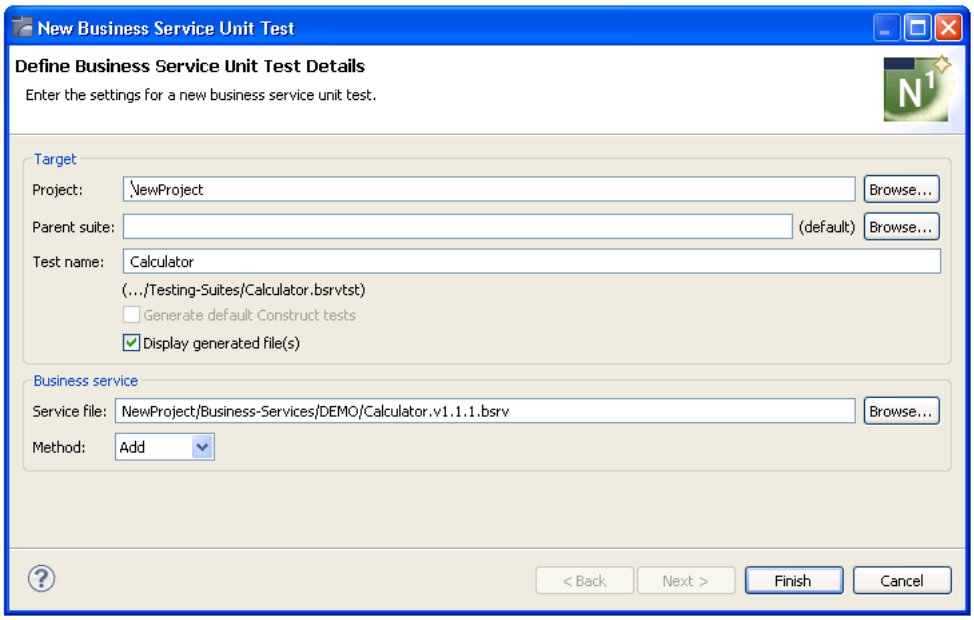
### Save as a Business Service Unit Test

After defining the input and output parameters for the test, you can save it as a business service unit test.

#### To save the test as a business service unit test

1 Select **Save As** on the **File** menu.

The **Define Business Service Unit Test Details** panel is displayed. For example:




2 Provide details for the unit test.

For information, see *Create a Unit Test for a Business Service*.

**Note:** You can create Ant scripts to run unit tests (file extension *.bsrvtst*, *.exttst*, *.nattst*, and *.seqtst*). For information, see *Creating Ant Scripts to Run Unit Tests*.

# Test a Subprogram Directly

This section describes how to test a subprogram directly. The following topics are covered:

- Access the Test Function
- Access the Debug Function
- Save as a Natural Unit Test

**Note:** The subprogram must be available locally. If the subprogram is not available locally, download it from the server.

## Access the Test Function

### To access the function to test a subprogram directly

1 Open the context menu for the subprogram in the **Navigator** view.

Or:

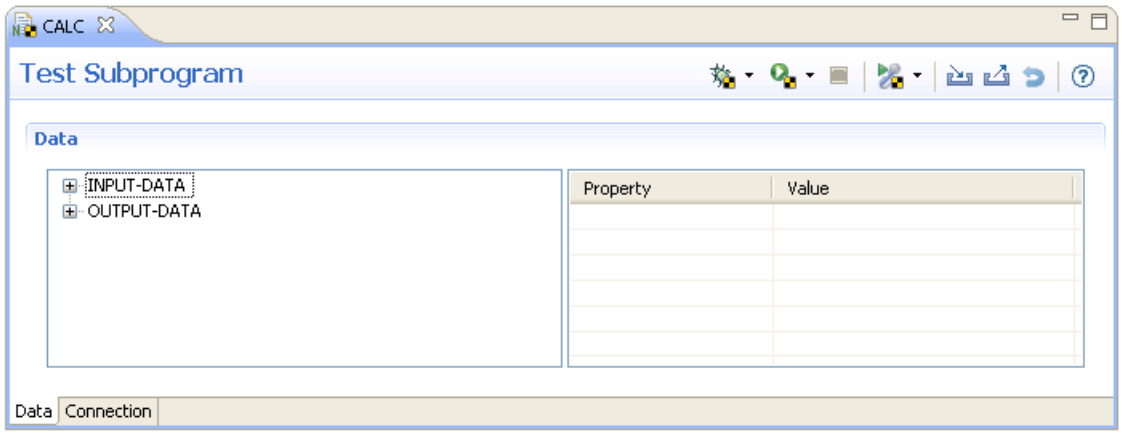
Open the context menu for the subprogram in the editor view.

Or:

Open the context menu for the subprogram in the **Dependencies** view.

2 Select **Testing > Test Subprogram**.

The subprogram is displayed in the editor view. For example:




This editor functions in the same way as the **Test Business Service** editor. The differences between this editor and the **Test Business Service** editor is that this editor includes the Debug

toolbar icon and the business service editor has an option to select the method (which can change which subprogram is tested internally).

**Note:** For information on using this editor, see *Features of the Test Editors* and *Test a Business Service Directly*.

### Access the Debug Function

This section describes how to access the Debug option from the **Test Subprogram** editor.

**Note:** To activate the Debug function, the **Use debug attach server** option must be selected in the Eclipse **Preferences > Software AG > Natural > Debug Attach Settings** window.

#### To access the Debug function

1 Open the context menu for the subprogram in the **Navigator** view.

Or:

Open the context menu for the subprogram in the editor view.

Or:

Open the context menu for the subprogram in the **Dependencies** view.

2 Select **Testing > Test Subprogram**.

The subprogram is displayed in the editor view.

3 Select on the editor toolbar to debug the subprogram using the values currently defined in the editor.

When a breakpoint is reached, the Debug perspective is displayed. For more information, see *Debug a Subprogram Directly*.

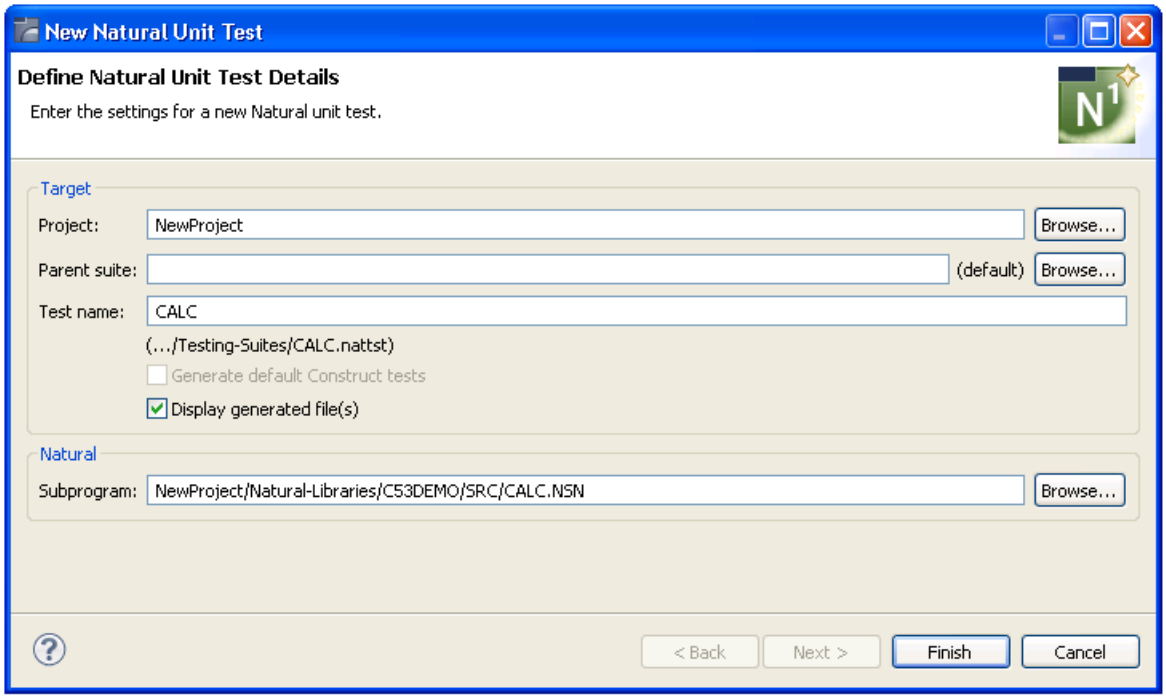
### Save as a Natural Unit Test

After defining the input and output parameters for the test, you can save it as a Natural unit test.

#### To save the test as a Natural unit test

1 Select **Save As** on the **File** menu.

The **Define Natural Unit Test Details** panel is displayed. For example:




2 Provide details for the unit test.

For information, see *Create a Unit Test for a Subprogram*.

**Note:** You can create Ant scripts to run unit tests (file extension *.bsrvtst*, *.exttst*, *.nattst*, and *.seqtst*). For information, see *Creating Ant Scripts to Run Unit Tests*.

## Debug a Subprogram Directly

This section describes how to debug a subprogram via RPC using the NaturalONE debugger and the values currently defined in the editor.

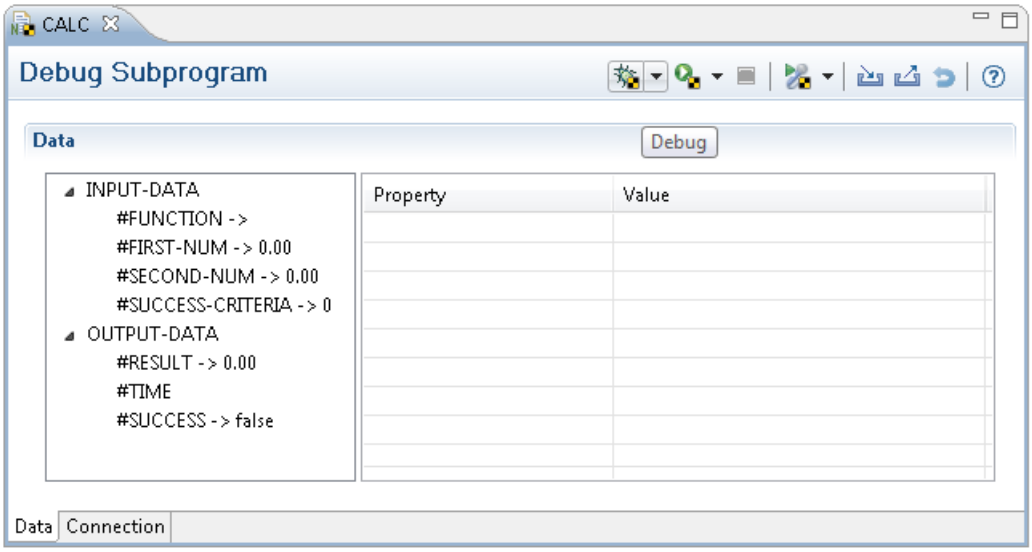
**Note:** To activate the **Debug** context menu, the **Use debug attach server** option must be selected in the Eclipse **Preferences > Software AG > Natural > Debug Attach Settings** window.

### To debug a subprogram

1 Open the context menu for the subprogram in the **Navigator** view.

2 Select **NaturalONE > Debug**.

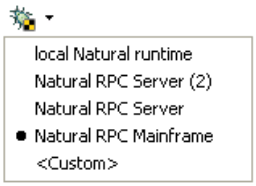
The subprogram is displayed in the editor view. For example:




This editor functions in the same way as the **Test Subprogram** editor.

3 Select the down arrow for to select the environment in which to debug the current file using the values defined in the editor.

For example:




4 Select to debug the current file using the values defined in the editor.

When a breakpoint is reached, the Debug perspective is displayed.

**Tip:** If you receive a WAIT timeout occurred error message, try increasing the timeout value for the selected RPC connection in the Eclipse **Preferences > Software AG > EntireX > RPC Environments** window. You can continue debugging in the Debug perspective when the editor receives the WAIT timeout error.

**Notes:**

1. For information on using this editor, see *Features of the Test Editors* and *Test a Subprogram Directly*.
2. For information on using a debug attach server, see *Using the Debugger* in *Using NaturalONE*.

# Export/Import Test Data

This section describes how to export and import test data for a business service and subprogram, which allows you to populate the test data quickly without re-entering each field name. These options are:

- Export test data (field names and values) from the test editor data tree to a new or existing test data file (extension *.tstdata*) in the workspace.

  **Note:** The *.tstdata* files can be stored anywhere in the workspace.

- Import an existing test data file in the workspace to the test editor (matching field names in the imported file to field names in the editor).

This section covers the following topics:


- Export Test Data
- Import Test Data


## Export Test Data

### To export test data to the workspace

1 Open the context menu for the business service (or subprogram) in the **Navigator** view.

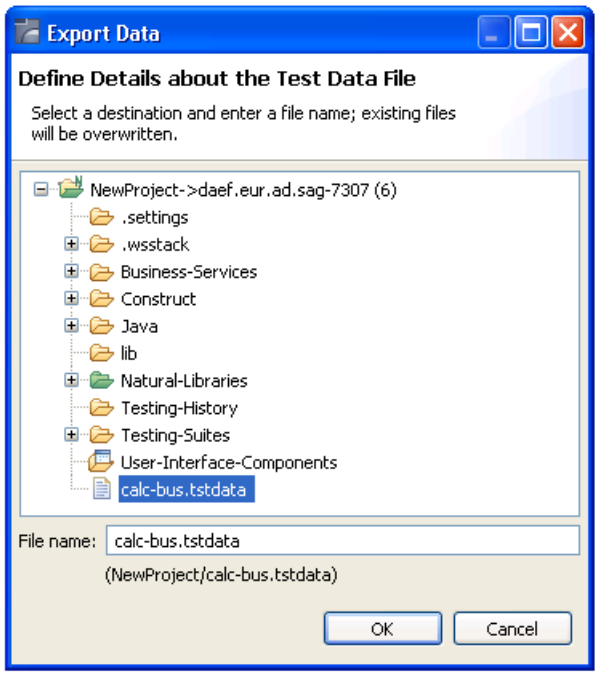
The testing options are displayed.

2 Select **Test Business Service** (or **Test Subprogram**).

The business service (or subprogram) is displayed in the editor view.

3 Select on the editor toolbar.

The **Define Details about the Test Data File** window is displayed. For example:




4 Select the location in which to export the test data file.

The last exported *.tstdata* file is selected.

**Note:** To overwrite data, select an existing file.

5 Type the name of the test data file in **File name**.

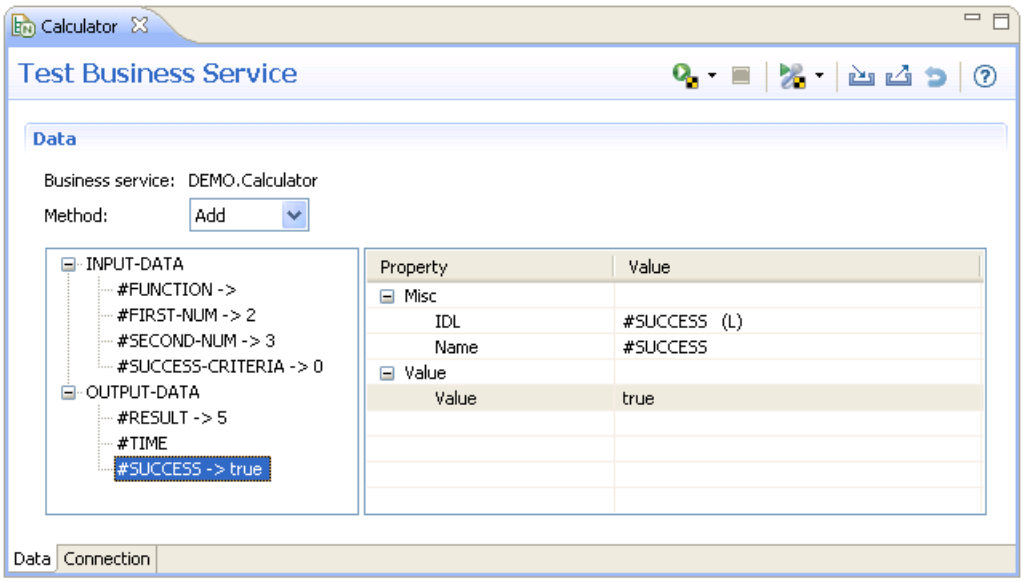
By default, the ".tstdata" extension is added to the file name.

6 Select **OK** to export the test data file.

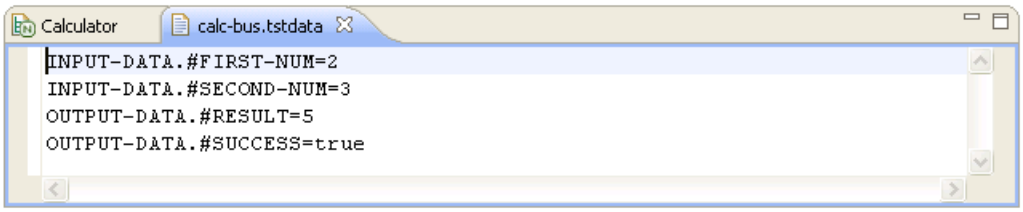
If the test data file currently exists (as shown in the example above), an overwrite confirmation dialog is displayed.

**Example**

The following example shows sample input for a business service test:




After exporting the data, the following test data (*.tstdata*) file is created:




You can modify this file using key=value pairs (for example, FIELDA=value). If the key begins with a hash character (#), then the field name must be preceded by a \ character (for example, \#FIELDB=value) or the field will be skipped. All other hash characters (such as CUSTOMER.#NAME=value) do not require the \ character.

**Tip:** Using this file as an example, you can create test data files for all the functions, save the files using appropriate names, and then change the values accordingly.

## Import Test Data

### To import test data from the workspace

1 Open the context menu for the business service (or subprogram) in the **Navigator** view.

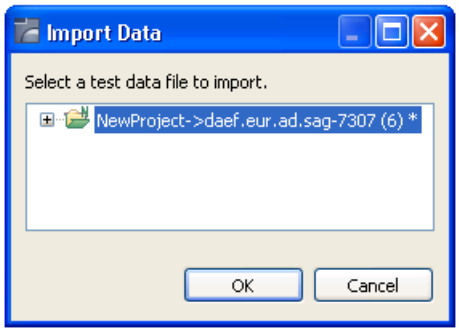
The testing options are displayed.

2 Select **Test Business Service** (or **Test Subprogram**).

The business service (or subprogram) is displayed in the editor view.

3 Select on the editor toolbar.

The **Import Data** window is displayed. For example:




4 Select the test data file to import (only projects/folders containing test data files are listed).

5 Select **OK** to import the file.

Any field in the imported test data file that does not have a matching field in the test editor tree, or has a matching field containing an invalid value, will not be imported and will not stop the import process. If this situation occurs, an Error log warning is displayed showing problem fields.

## Export Test Data to a CSV File

This section describes how to record the data used to test a business service or subprogram directly and then export it to a CSV file (file extension *.csv*). You can then use this file as input to create an external data unit test. For information, see *Create an External Data Unit Test*.

### To record the test data and export it to a CSV file

1 Open the context menu for the business service (or subprogram) in the **Navigator** view.

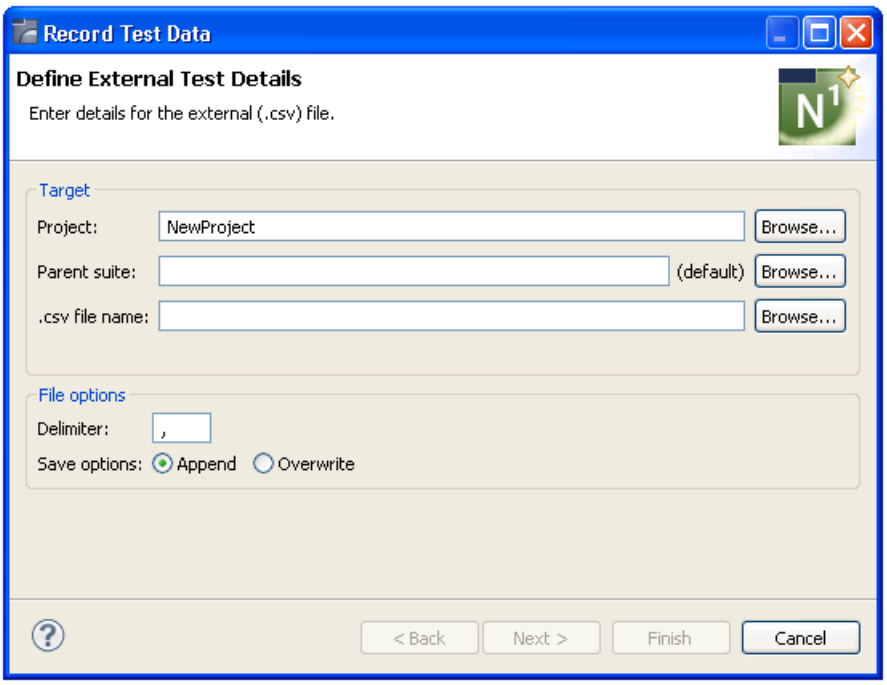
The testing options are displayed.

2 Select **Test Business Service** (or **Test Subprogram**).

The business service (or subprogram) is displayed in the editor view.

3 Select on the NaturalONE toolbar to begin recording.

The **Define External Test Details** panel is displayed. For example:




4 Type the name of the external data file in **.csv file name** or select **Browse** to display a window listing the available files for selection.

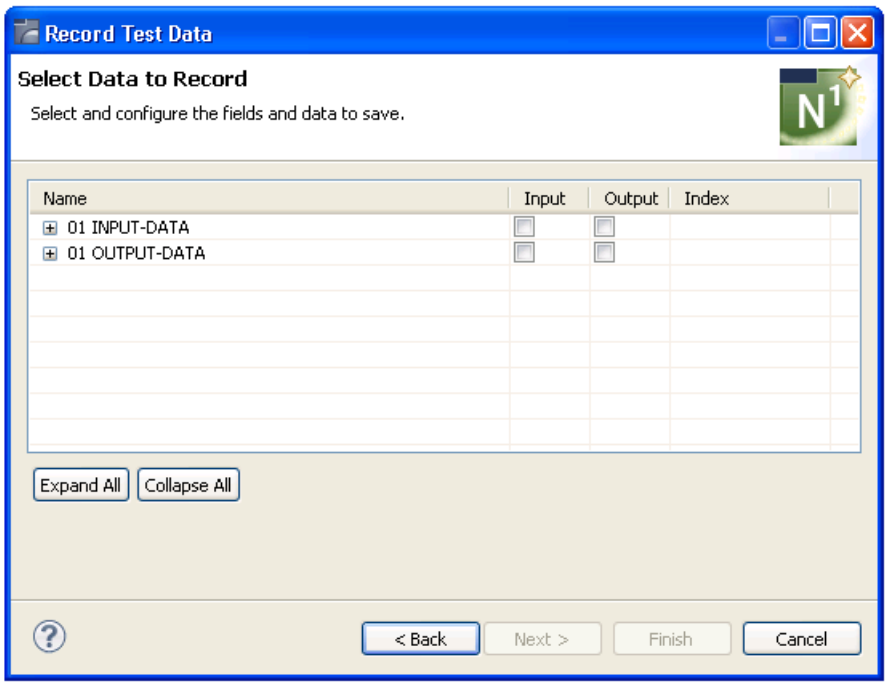
Optionally, you can use the **Define External Test Details** panel to:

| Task | Procedure |
|---|---|
| Change the name of the project in which to create the external data file. | Type the name of the Natural project in **Project** or select **Browse** to display a window listing the existing projects for selection.<br><br>**Note:** The project must currently exist. |
| Provide the name(s) of a subfolder(s) in which to save the external data file. If the folder does not currently exist, it will be created for you. | Type the name of the folder in **Parent suite** or select **Browse** to display a window listing the available folders for selection.<br><br>By default, the external data file is stored in the **Testing-Suites** folder in the current project. If you specify a suite folder name, it becomes a subfolder in the **Testing-Suites** folder and the file will be stored in that folder. |
| Change the delimiter character used to separate input values in the external data file you are generating. | Type the character in **Delimiter**. |
| Replace test data in an existing CSV file (file extension *.csv*) with new data. | Select "Overwrite" in **Save options**.<br><br>**Note:** If you specify the name of an existing file in **.csv file name** and the **Save options** is "Append" (default), the test |

| Task | Procedure |
|---|---|
| | data is appended to existing test data in the file. If the mode is "Overwrite", existing test data in the file will be overwritten. |

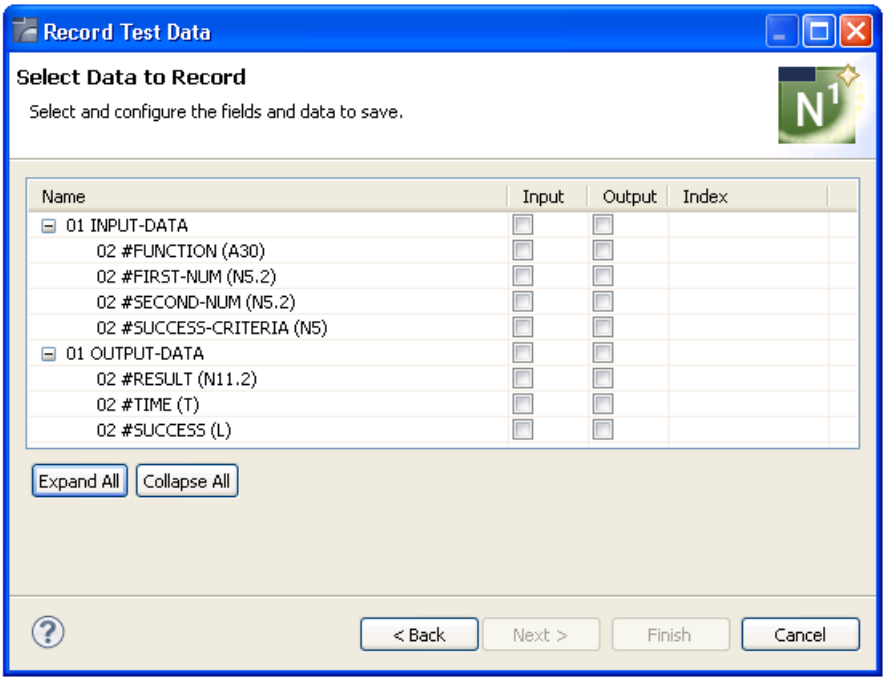
5 Select **Next**.

The **Select Data to Record** panel is displayed. For example:




6 Select **Expand All**.

The level 1 fields are expanded to display the lower level fields. For example:




**Note:** To collapse the fields, select **Collapse All**.

7 Select **Input** and/or **Output** for each level 1 field you want to include in the recording.

Only the selected data for each field will be saved.

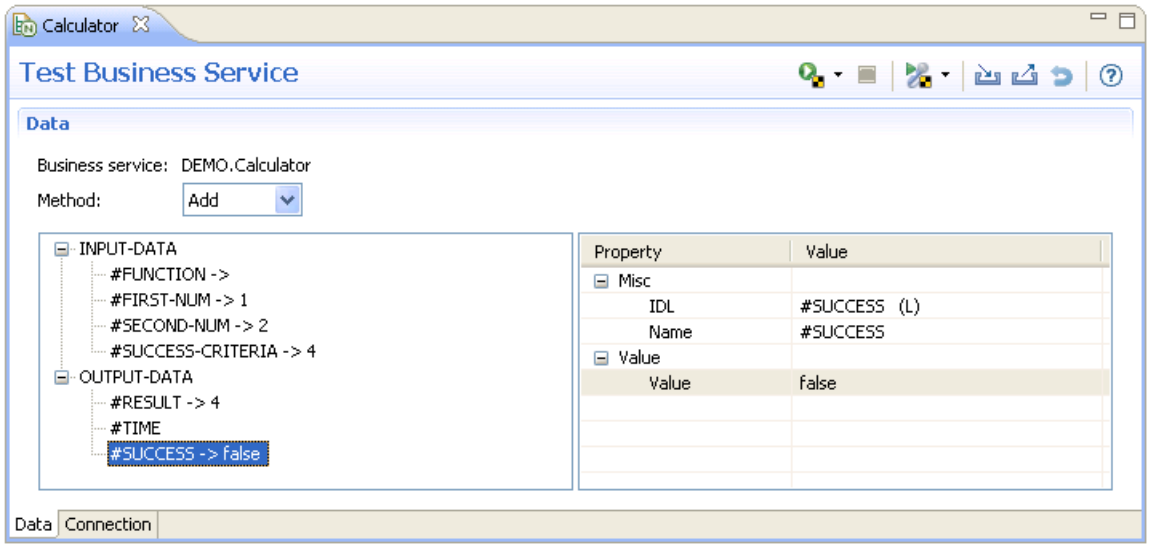
8 Provide index values in **Index** for any array fields.

9 Select **Finish** to begin recording.

The **Recording** icon changes to on the toolbar.

10 Define the test data in the editor view.

For example:




**Note:** For information on using this editor, see *Features of the Test Editors* and *Test a Business Service Directly*.

11 Select to run the test.

Repeat steps 10 and 11 for each test containing data you want to record.

12 Select to stop recording.

The generated CSV file is displayed in the **Testing-Suites** node in the **Navigator** view.

# 6 Create a Unit Test for a Business Service or Subprogram


- Enable for Application Testing ..... 44
- Create a Unit Test for a Business Service ..... 45
- Create a Unit Test for a Subprogram ..... 63
- Generate Default Unit Tests ..... 66
- Create a New Unit Test Suite ..... 76
- Create Summary Reports for Unit Test Log Files ..... 78
- Use the Dependencies View ..... 83

This section describes how to create a Natural unit test for a business service or subprogram. This allows you to specify a business service or subprogram to test, supply input values, and then provide validation criteria for the output of the call (for example, you can supply two numbers as the input values and a value for the result field as the validation criteria).

**Note:** You can create Ant scripts to run unit tests (file extension *.bsrvtst*, *.exttst*, *.nattst*, and *.seqtst*). For information, see ***Creating Ant Scripts to Run Unit Tests***.

## Enable for Application Testing

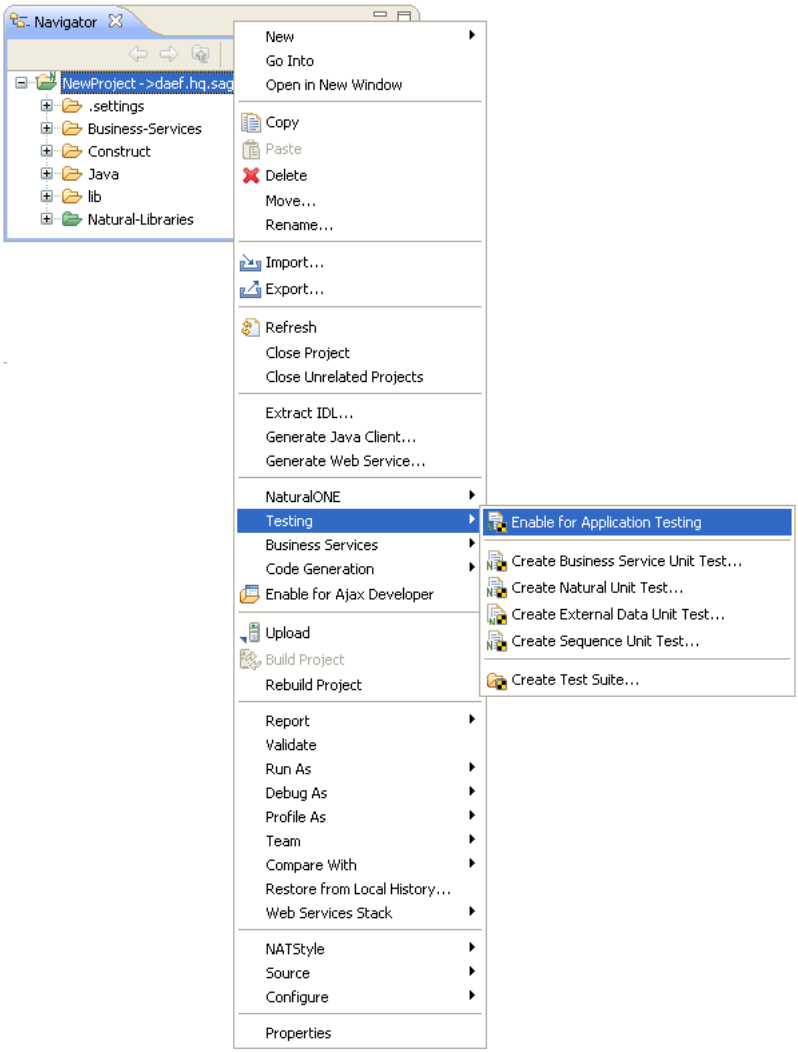
When you create a new unit test, the Natural project containing the test is automatically enabled for application testing. This will create the **Testing-Suites** folder in the **Navigator** view and provide warning markers on existing unit test files that are not in the **Testing-Suites** folder or its subfolders. This section describes how to manually enable a Natural project for application testing.

### To enable a Natural project for application testing

1 Open the context menu in the **Navigator** view for the Natural project containing the business service or subprogram you want to test.

2 Select **Testing > Enable for Application Testing**.

For example:




The **Testing-Suites** folder is added to the project. All new unit tests will be generated into this folder (or subfolder).

## Create a Unit Test for a Business Service

This section describes how to create a unit test for a business service. The following topics are covered:

- Create the Unit Test
- Configure Input Parameters
- Define Validations
- Run the Unit Test
- Open a Previous Unit Test
- Run a Unit Test in Another Environment

- Test for an Expected Error
- Test an Array Field

## Create the Unit Test

### To create a unit test for a business service

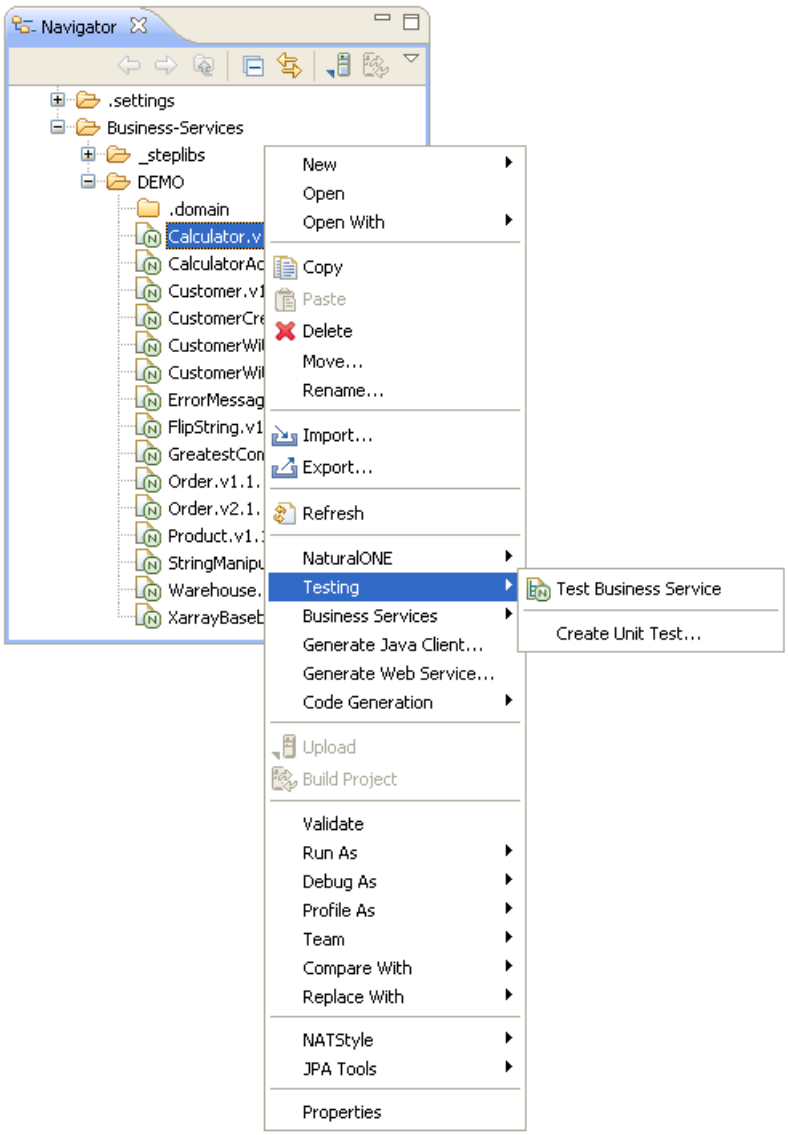
1 Open the context menu for the Natural project containing the business service in the **Navigator** view.

Or:

Open the context menu for the business service in the **Navigator** view.

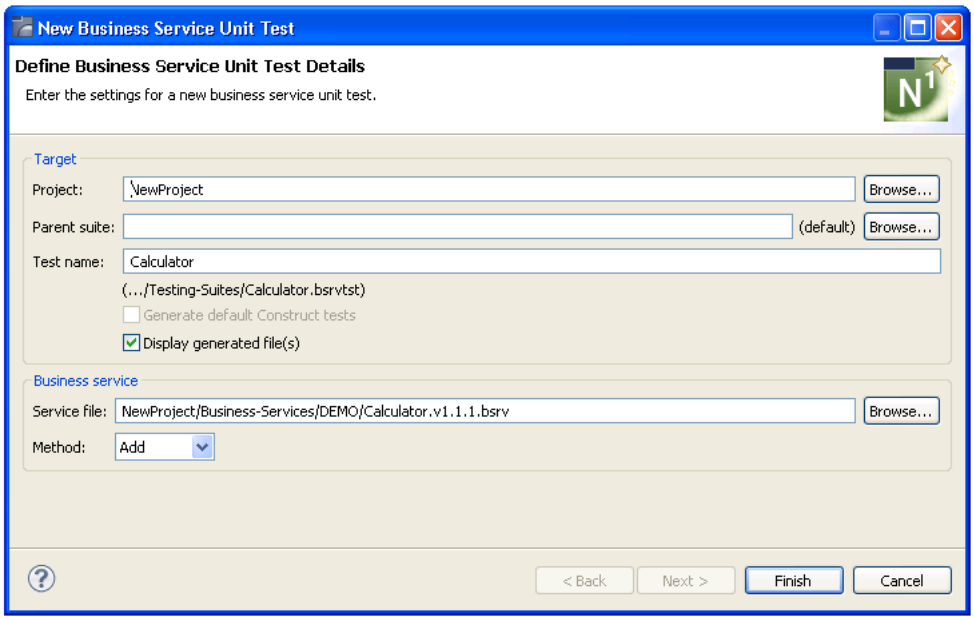
2 Select **Testing**.

The test options for business services are displayed. For example:




3 Select **Create Unit Test**.

The **Define Business Service Unit Test Details** panel is displayed. For example:



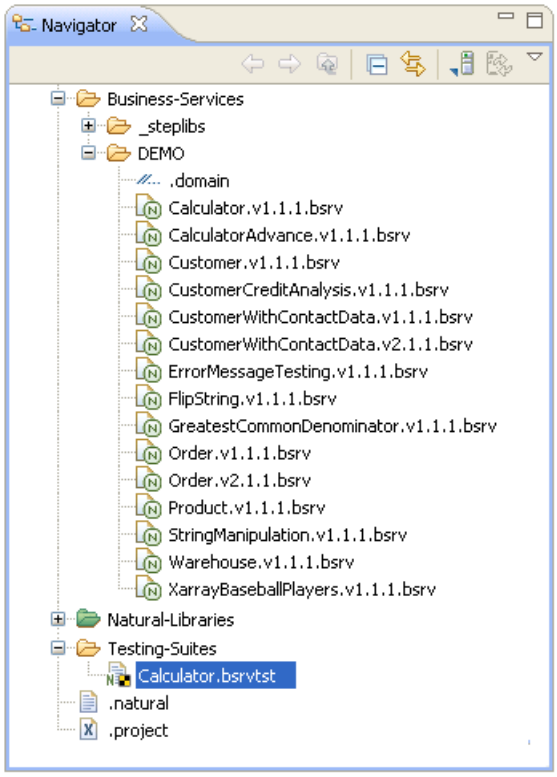

Using this panel, you can:

| Task | Procedure |
|---|---|
| Change the name of the project in which to create the unit test. | Type the name of the Natural project in **Project** or select **Browse** to display a window listing the existing projects for selection.<br><br>**Note:** The project must currently exist. |
| Provide the name(s) of a subfolder(s) in which to save the unit test. If the folder does not currently exist, it will be created for you. | Type the name of the folder in **Parent suite** or select **Browse** to display a window listing the available folders for selection.<br><br>By default, the unit test is stored in the **Testing-Suites** folder in the current project. If you specify a suite folder name, it becomes a subfolder in the **Testing-Suites** folder and the unit test will be stored in that folder. |
| Change the default name for the unit test. | Type a new name in **Test name**. File names are saved with the *.bsrvtst* extension. |
| Generate default unit tests for object-maintenance functions and/or object-browse keys defined for business service subprograms. | Select **Generate default Construct tests**. This option is enabled when the unit test will be created for a business service that uses Velocity or Construct-generated object-browse or object-maintenance subprograms. For information, see *Generate Default Unit Tests*. |
| Not display the generated files in the editor view after generation. | Deselect **Display generated file(s)**. |
| Change the location of the folder containing the business service file. | Type or select a new folder in **Service file**. |

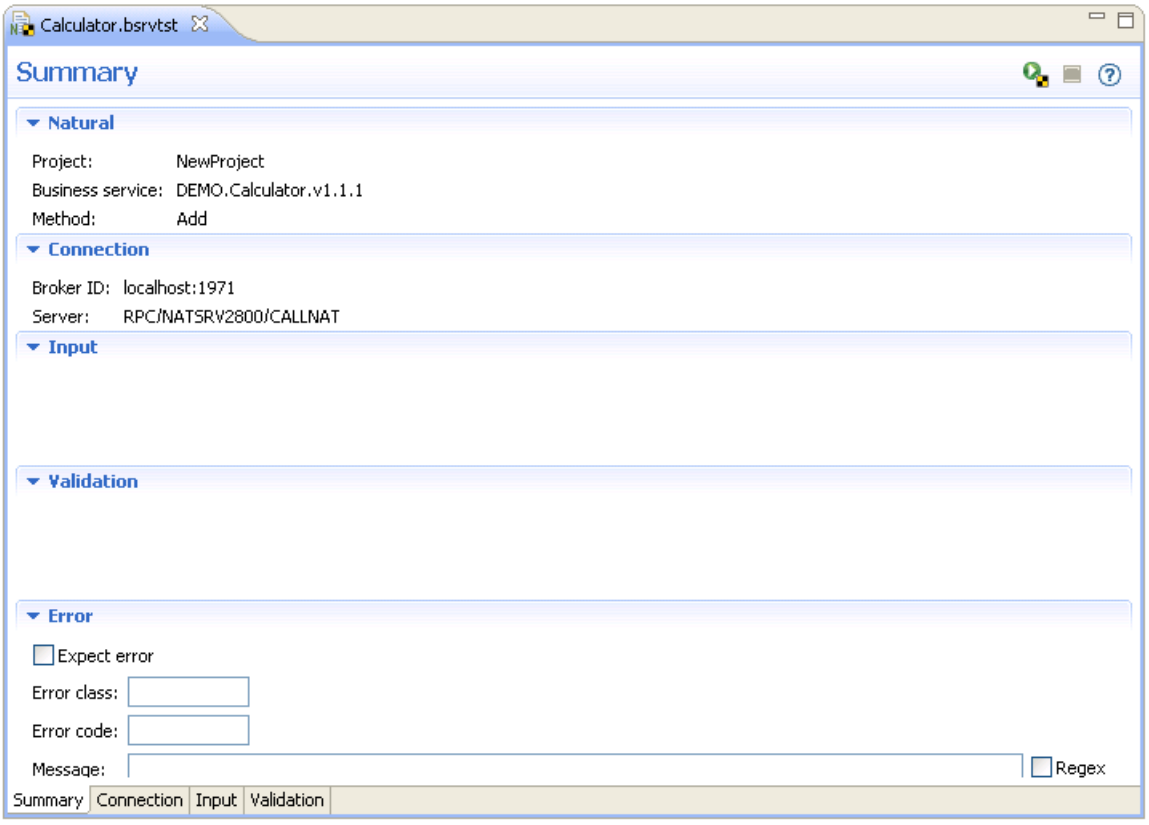
| Task | Procedure |
|---|---|
| Select a different method to test. | Select the method in **Method**. |

4 Select **Finish**.

The unit test is displayed in the **Testing-Suites** folder in the **Navigator** view. For example:




The test is also displayed in the editor view. For example:




The **Summary** tab displays information about the test, such as the name of the project, business service, and method. It also displays the default connection settings. To define another connection in which to run the test, see *Define Connections*.

**Note:** You can use this tab to define an expected error, which allows a test to pass whenever the expected error occurs. You can also use the tab to search for specified text in an error message. For information, see *Test for an Expected Error*.

5 Select the **Input** tab and define which input parameters are sent to the server.

For information, see *Configure Input Parameters*.

6 Select the **Validation** tab and define which values must be returned for a successful test.

For information, see *Define Validations*.

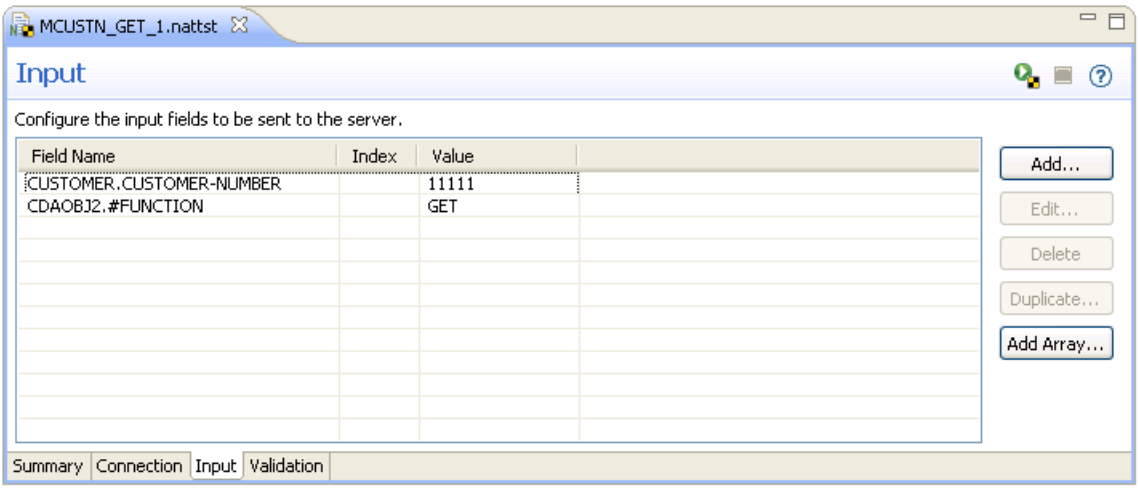
**Note:** You can create Ant scripts to run business service unit tests (file extension *.bsrvtst*). For information, see *Creating Ant Scripts to Run Unit Tests*.

## Configure Input Parameters

### To configure the input parameters sent to the server

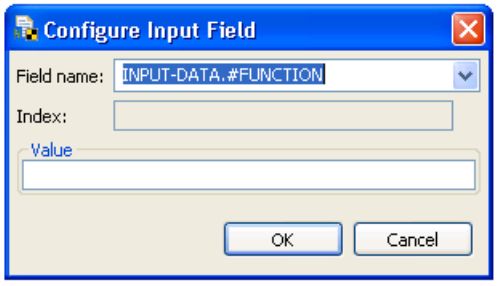
1 Select the **Input** tab in the unit test editor.

For example:




2 Select **Add**.

The **Configure Input Field** window is displayed. For example:




The list of available controls in **Field name** is based on the data type of the input field. If you selected a logical field, for example, two option buttons are displayed to select "true" or "false". If the field is an array, you can type the index for the array in **Index**.

3 Select the name of the input field in **Field name**.

4 Type the field value in **Value**.

5 Select **OK**.

The new field is added to the list of fields on the **Input** tab.

Optionally, you can use the **Input** tab to:

| Task | Procedure |
|---|---|
| Edit an input field. | See *Edit an Input Field*. |
| Remove one or more input fields. | Select one or more input fields in **Field Name** using standard selection techniques and select **Delete**. The field(s) is removed from the list of fields and will not be sent to the server. |
| Duplicate an input field. | See *Duplicate an Input Field*. |
| Add multiple elements to an array field. | See *Add Multiple Elements for an Array Field*. This option is enabled when the PDA contains array fields. |

### Edit an Input Field

#### To edit an input field

1 Select the input field in **Field Name** on the **Input** tab.

2 Select **Edit**.

The **Configure Input Field** window is displayed to edit the field information.

3 Select **OK** to save the changes.

Or:

Select the input field in **Field Name** and edit the **Value** and/or **Index** values within the table.

### Duplicate an Input Field

#### To duplicate an input field

1 Select the input field in **Field Name** on the **Input** tab.

2 Select **Duplicate**.

The **Configure Input Field** window is displayed to edit the field information.

3 Select **OK** to save the duplicate field.

### Add Multiple Elements for an Array Field

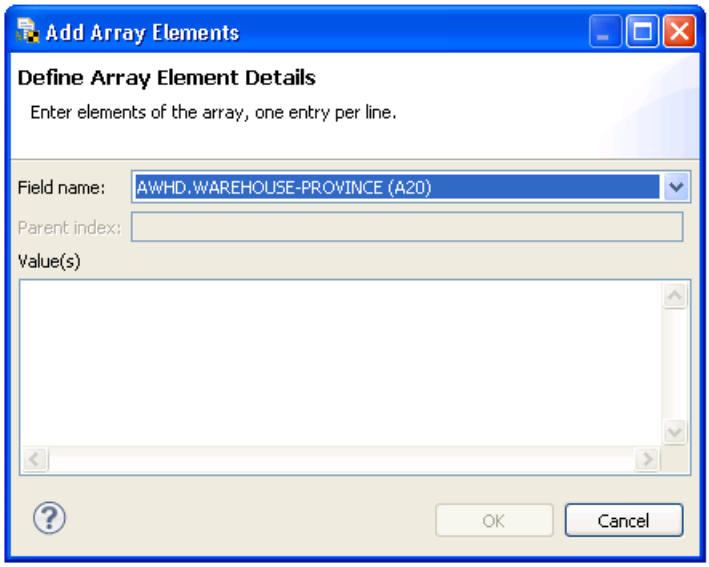
This section describes how to define a range of values for an array field.

**Note:** The **Add Array** option does not support byte array and date/time fields.

#### To add multiple elements to an array field at the same time

1 Select **Add Array**.

The **Define Array Element Details** window is displayed. For example:




2 Type each element of the array in **Value(s)**, one entry per line.

3 Select **OK** to save the array field.

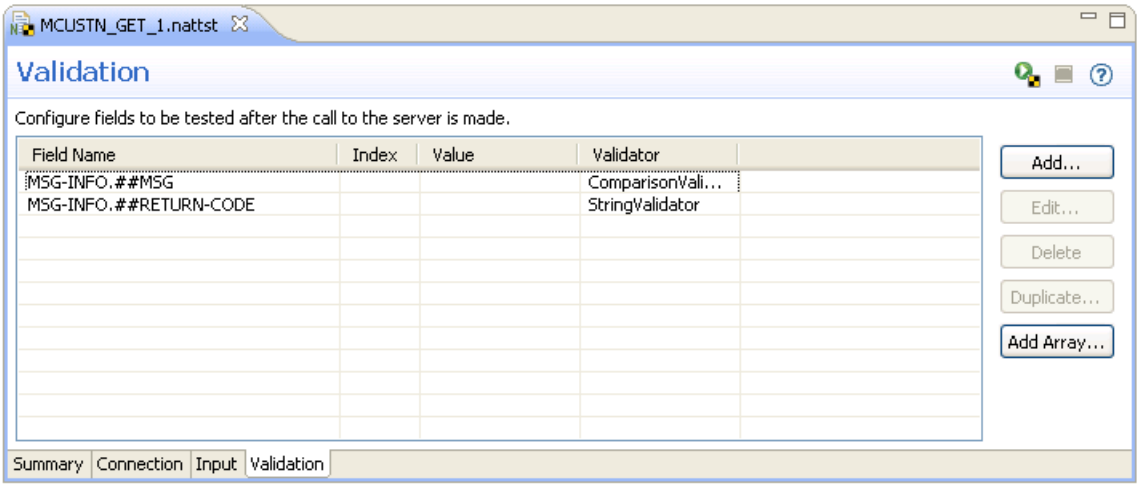
## Define Validations

This section describes how to create unit test validations for Natural errors and data entry based on validator types (i.e., not restricted to characters in the data type).

#### To define validations

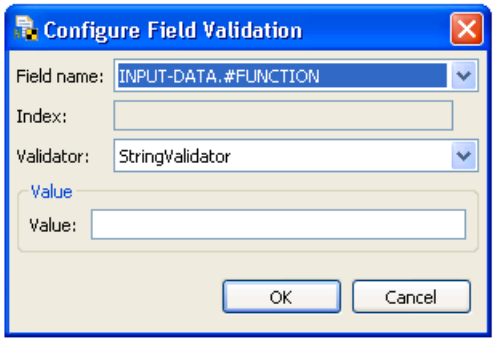
1 Select the **Validation** tab in the business service unit test editor.

For example:




2 Select **Add**.

The **Configure Field Validation** window is displayed. For example:




The list of available controls in **Field name** is based on the data type of the input field. If you select a logical field, for example, two option buttons are displayed to select "true" or "false". If the field is an array, you can type the index for the array in **Index**.

3 Select the name of the input field in **Field name**.

4 Select the type of validator to use for the input field in **Validator**.

The type of validator to use depends on the type of data in the field. The available validators are:

- BooleanValidator
- ByteValidator
- ComparisonValidator (displays a combo box with the options: ">", "<", "=", "<=", ">=")
- DateValidator

- DecimalValidator
- IntegerValidator
- RegexValidator (creates regular expressions to validate the contents of a field)
- StringValidator
- TimeValidator

5 Select **OK**.

The new field is added to the list of fields on the **Validation** tab.

Optionally, you can use the **Validation** tab to:

| Task | Procedure |
|---|---|
| Edit a field validation. | See *Edit a Field Validation*. |
| Remove one or more field validations. | Select one or more fields in **Field Name** using standard selection techniques and select **Delete**. The field validation(s) is removed. |
| Duplicate a field validation. | See *Duplicate a Field Validation*. |
| Add multiple validations for an array field. | See *Add Multiple Validations for an Array Field*. This option is enabled when the PDA contains array fields. |

### Edit a Field Validation

#### To edit a field validation

1 Select the field in **Field Name** on the **Validation** tab.

2 Select **Edit**.

The **Configure Field Validation** window is displayed to edit the field information.

3 Select **OK** to save the changes.

Or:

Select the input field in **Field Name** and edit the **Value** and/or **Index** values within the table.

### Duplicate a Field Validation

#### To duplicate a field validation

1 Select the input field in **Field Name** on the **Input** tab.

2 Select **Duplicate**.

The **Configure Field Validation** window is displayed to edit the information.

3 Select **OK** to save the duplicate field validation.

### Add Multiple Validations for an Array Field

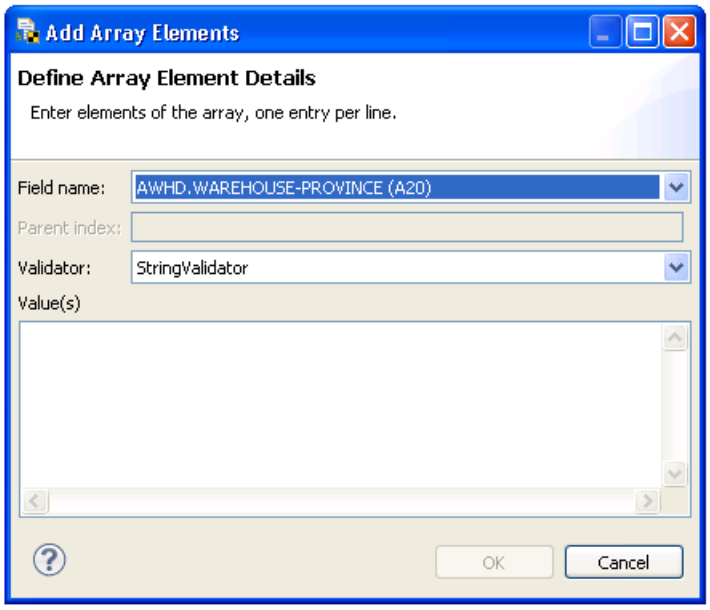
This section describes how to define validations for an array field.

**Note:** The **Add Array** option does not support byte array and date/time fields.

#### To add multiple validations to an array field

1 Select **Add Array**.

The **Define Array Element Details** window is displayed. For example:




2 Type each element of the array in **Value(s)**, one entry per line.

3 Select **OK** to save the array field.

## Run the Unit Test

This section describes how to run one or more unit tests. It includes information about the **Natural Unit Test** window.

**Note:** You can create Ant scripts to run unit tests (file extension *.bsrvtst*, *.exttst*, *.nattst*, and *.seqtst*). For information, see *Creating Ant Scripts to Run Unit Tests*.

### To run one or more unit tests

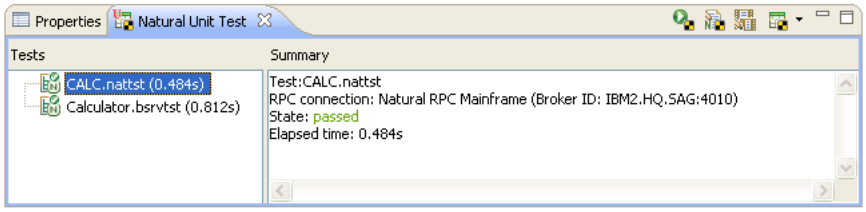
1 Open the context menu for one of the following items in the **Navigator** view.

- A project containing the **Testing-Suites** folder.
- The **Testing-Suites** folder or a subfolder within the folder.
- One or more unit test files (file extension *.nattst* or *.bsrvtst*), regardless of where they exist. Use standard selection techniques to open the unit test(s). Any test files stored outside of the **Testing-Suites** folder display a warning marker in the **Navigator** view and an entry in the **Problems** view indicating that they are not in the proper place.

2 Select **Testing > Run Unit Test(s)**.

**Note:** You can also use the context menu to change the environment in which a test is run. For information, see *Run a Unit Test in Another Environment*.

The selected tests are displayed in the editor view and the results of the test are displayed in the **Natural Unit Test** view. For example:

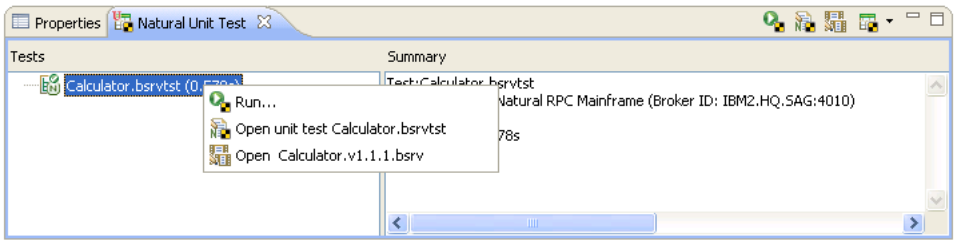



**Note:** If the test did not pass, a red circle ( ) is displayed on the test icon in the **Tests** section and **State: failed** is displayed in the **Summary** section.

The following table describes each of the options available on the toolbar for the **Natural Unit Test** view:

| Toolbar Icon | Description |
|---|---|
| | Runs the current unit test using the values defined in the editor view.<br><br>**Tip:** You can also select in the editor view to run the test. |
| | Selects the current unit test in the editor view. |
| | Opens the business service or Natural subprogram used for the current unit test in the editor view. |
| | Displays the test history for the last 10 unit tests that were run during the current Eclipse session and allows you to select a previous test and load it into the editor. For information, see *Open a Previous Unit Test*. |

The **Tests** section in the **Natural Unit Test** view displays the name of each unit tests that have been run. You can use the context menu for a unit test in the **Tests** section to select more options. For example:



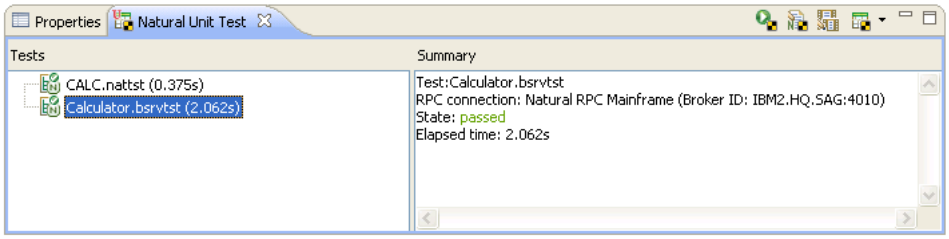

Using this menu, you can:

| Task | Procedure |
|---|---|
| Run the unit test. | Select **Run**. |
| Open the unit test file in the editor view. | Select **Open unit test** *UnitTestName*. The following file types are available for selection:<br><br>■ business service (file extension *.bsrvtst*)<br>■ external data (file extension *.exttst*)<br>■ Natural unit test (file extension *.nattst*)<br>■ sequence (file extension *.seqtst*) |
| Open the associated business service or Natural subprogram file in the editor view. | Select **Open** *BusinessServiceName*.**bsrv** or **Open** *NaturalSubprogramName*.**NSN**. The following file types are available for selection:<br><br>■ business service (file extension *.bsrv*)<br>■ external data (file extension *.NSN*)<br><br>**Note:** This option is not available for external data or sequence unit tests. |

The **Summary** section in the **Natural Unit Test** view displays:

- Name of the test
- Name of the RPC connection
- Whether the test passed or failed
- Length of time in seconds that the unit test executed before completing

To see the results of another test, select the test in the **Tests** section and the results are displayed in the **Summary** section. For example:

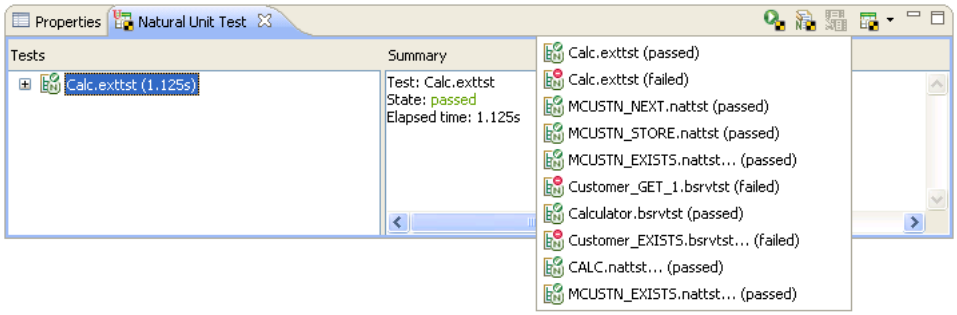



## Open a Previous Unit Test

### To open a previous unit test

1 Select ▾ on the toolbar in the **Natural Unit Test** view.

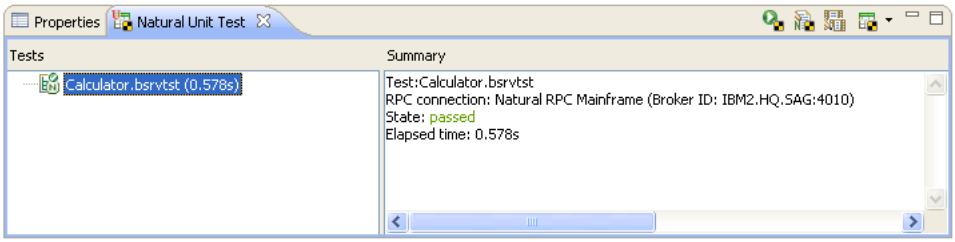
A list of the last 10 tests run during the current Eclipse session is displayed with a message indicating the success of each test. For example:




In this example, seven tests passed and three tests failed.

2 Select the test you want to open.

The test is displayed in the **Natural Unit Test** view. For example:




3 Open the context menu for the test.

4 Select the unit test file in **Open unit test** *UnitTestName*.*nnn***tst**.

The following unit test file types are available:

- business service (file extension *.bsrvtst*)
- external data (file extension *.exttst*)
- Natural unit test (file extension *.nattst*)
- sequence (file extension *.seqtst*)

## Run a Unit Test in Another Environment

You can run any unit test in another environment.

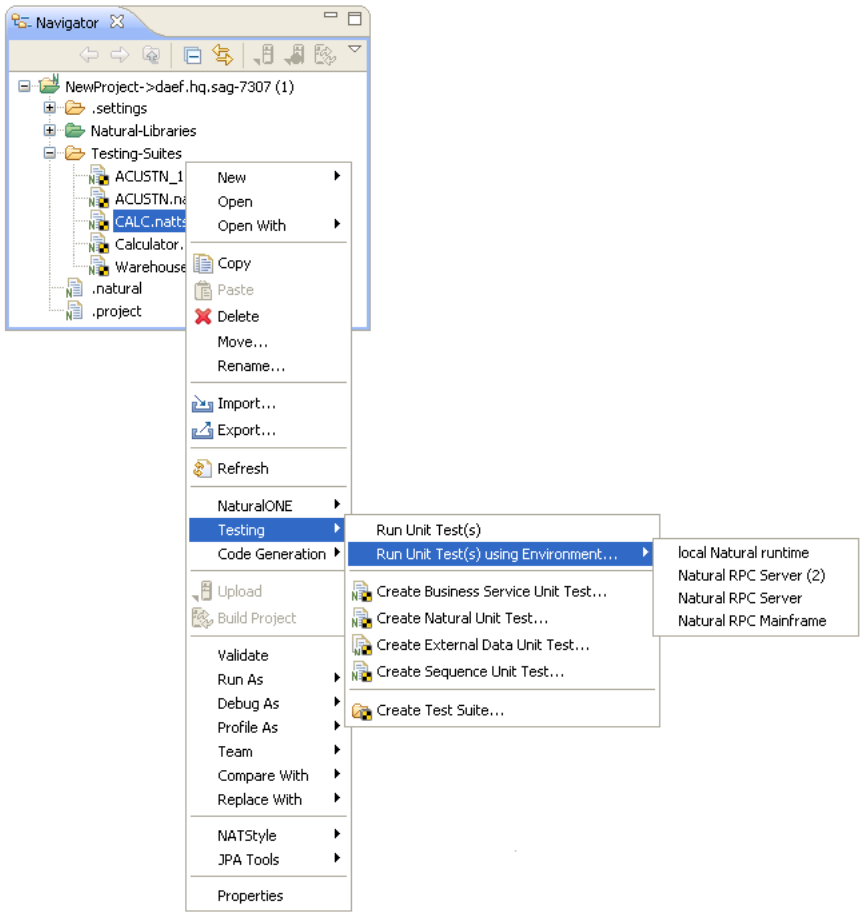
### To run a unit test in another environment

1 Open the context menu for one of the following items in the **Navigator** view.

- A project containing the **Testing-Suites** folder.
- The **Testing-Suites** folder or a subfolder within the folder.
- One or more unit test files (file extension *.bsrvtst*, *.exttst*, *.nattst*, and *.seqtst*), regardless of where they exist.

2 Select **Testing > Run Unit Test(s) using Environment**.

For example:



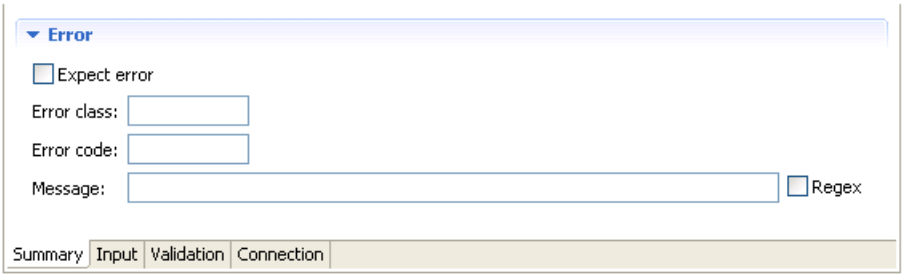

3 Select the environment in which you want to run the test.

The results of the test are displayed in the **Natural Unit Test** view.

**Note:** The list of environments is defined in the **Preferences** window for RPC environments. For information on adding additional environments to the list, see *Define Additional RPC Environments*.

### Test for an Expected Error

To allow a test to pass with an expected error, define information about the error in the **Error** section of the **Summary** tab. For example:




This will allow a test to fail only if it encounters an unexpected error.

### To test for an expected error

1 Select **Expect error**.

2 Type the error class in **Error class**.

For Natural errors, the error class is 1014.

3 Type the error code in **Error code**.

You can also use the **Error** section to search the message text for a specific string.

### To search the message text for a specified string

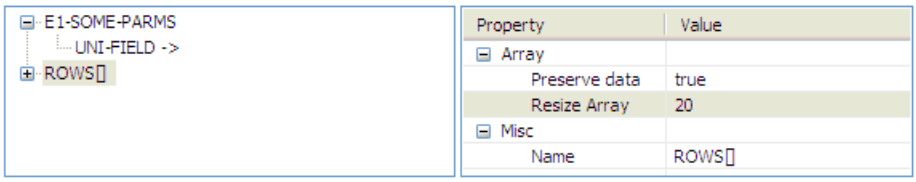
1 Type the string in **Message**.

2 Select **Regex**.

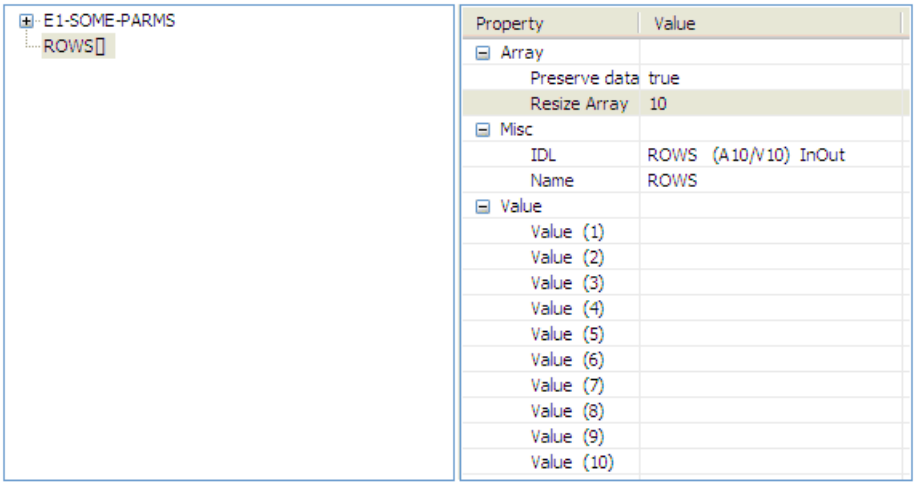
If you specify ".* division by zero.*", for example, Regex will search for "division by zero" anywhere in the error message.

## Test an Array Field

You can expand or reduce an X-array using the `Resize Array` property. For example:




For some arrays, all values are displayed. For example:




## Create a Unit Test for a Subprogram

### To create a unit test for a subprogram

1 Open the context menu for the Natural project containing the subprogram in the **Navigator** view.

Or:

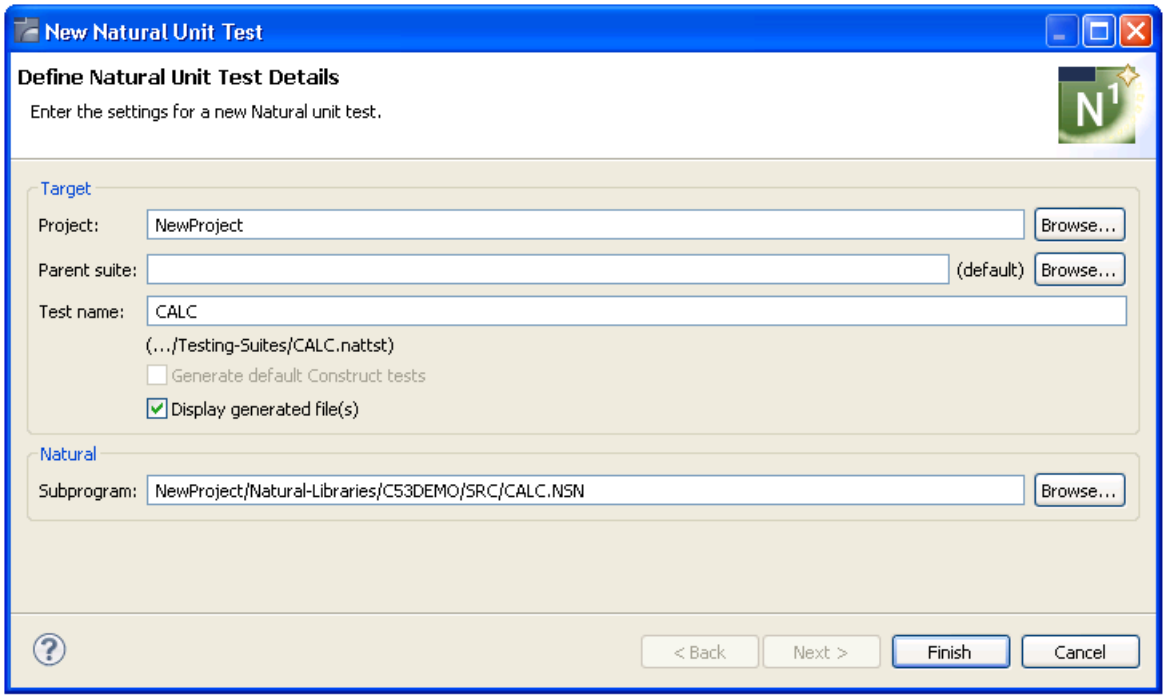
Open the context menu for the subprogram in the **Navigator** view.

2 Select **Testing**.

The test options for subprograms are displayed.

3 Select **Create Unit Test**.

The **Define Natural Unit Test Details** panel is displayed. For example:



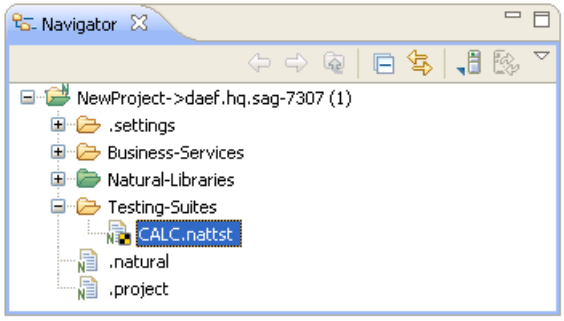

Using this panel, you can:

| Task | Procedure |
|---|---|
| Change the name of the project in which to create the unit test. | Type the name of the Natural project in **Project** or select **Browse** to display a window listing the existing projects for selection.<br><br>**Note:** The project must currently exist. |
| Provide the name(s) of a subfolder(s) in which to save the unit test. If the folder does not currently exist, it will be created for you. | Type the name of the folder in **Parent suite** or select **Browse** to display a window listing the available folders for selection.<br><br>By default, the unit test is stored in the **Testing-Suites** folder in the current project. If you specify a suite folder name, it becomes a subfolder in the **Testing-Suites** folder and the unit test will be stored in that folder. |
| Change the default name for the unit test. | Type a new name in **Test name**. File names are saved with the *.bsrvtst* extension. |
| Generate default unit tests for object-maintenance functions and/or object-browse keys defined for Natural subprograms. | Select **Generate default Construct tests**. This option is enabled when the unit test will be created for Velocity or Construct-generated object-browse or object-maintenance subprograms. For information, see *Generate Default Unit Tests*. |
| Not display the generated files in the editor view after generation. | Deselect **Display generated file(s)**. |

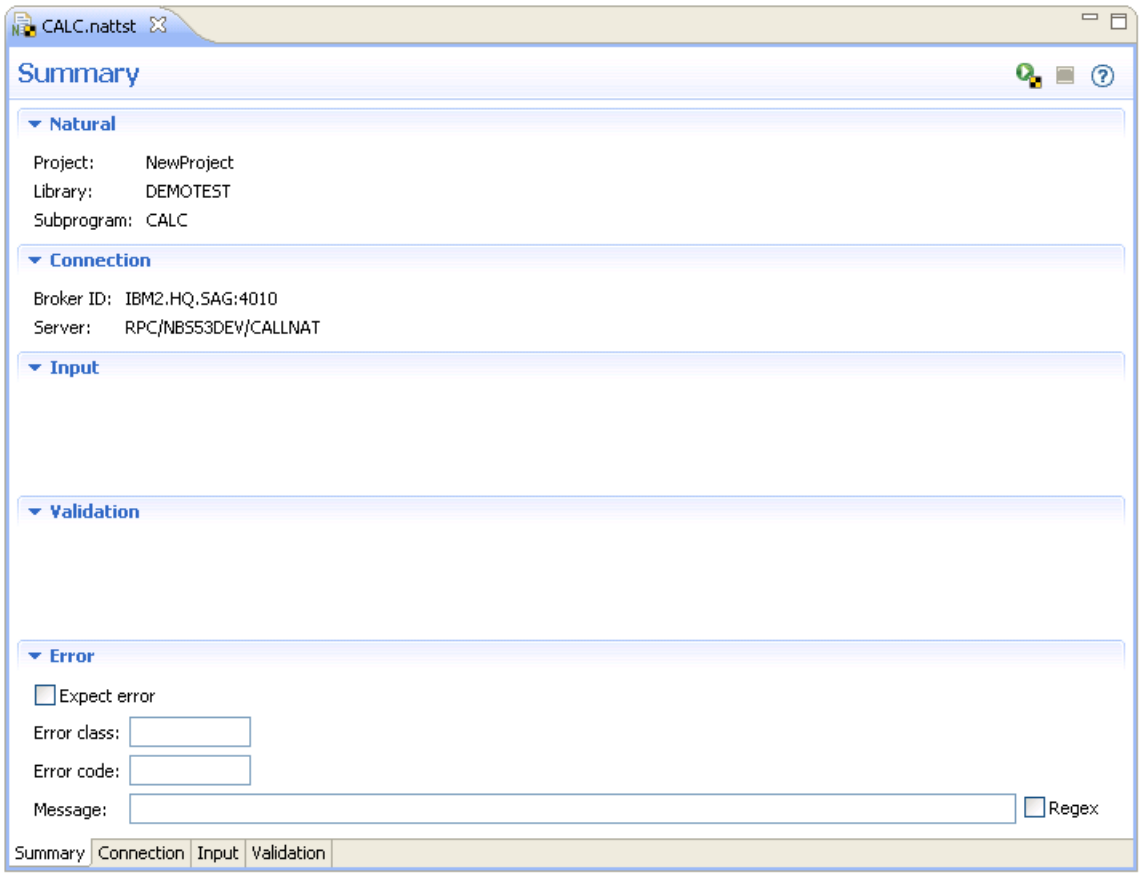
| Task | Procedure |
|---|---|
| Change the location of the folder containing the subprogram file. | Type or select a new folder in **Subprogram**. |

4 Select **Finish**.

The unit test is displayed in the **Testing-Suites** folder in the **Navigator** view. For example:




The test is also displayed in the editor view. For example:



The **Summary** tab displays information about the test, such as the name of the project, library, and subprogram. It also displays the default connection settings. To define another connection in which to run the test, see *Define Connections*.

**Note:** You can use this tab to define an expected error, which allows a test to pass when the expected error occurs. You can also use the tab to search for specified text in an error message. For information, see *Test for an Expected Error*.

5 Select the **Input** tab and define which input parameters are sent to the server.

For information, see *Configure Input Parameters*.

6 Select the **Validation** tab and define which values must be returned for a successful test.

For information, see *Define Validations*.

7 Run the test.

For information, see *Run the Unit Test*.

**Note:** You can create Ant scripts to run Natural unit tests (file extension *.nattst*). For information, see *Creating Ant Scripts to Run Unit Tests*.

## Generate Default Unit Tests

This section describes how to generate default unit tests for browse keys and maintenance functions (for example, GET, NEXT, etc.) defined for Velocity or Construct-generated object-browse or object-maintenance subprograms. If a business service uses both object-browse and object-maintenance subprograms, default tests can be generated for both the browse keys and the maintenance functions.

This section covers the following topics:

- Generate Tests for a Business Service

- Generate Tests for a Natural Subprogram

## Generate Tests for a Business Service

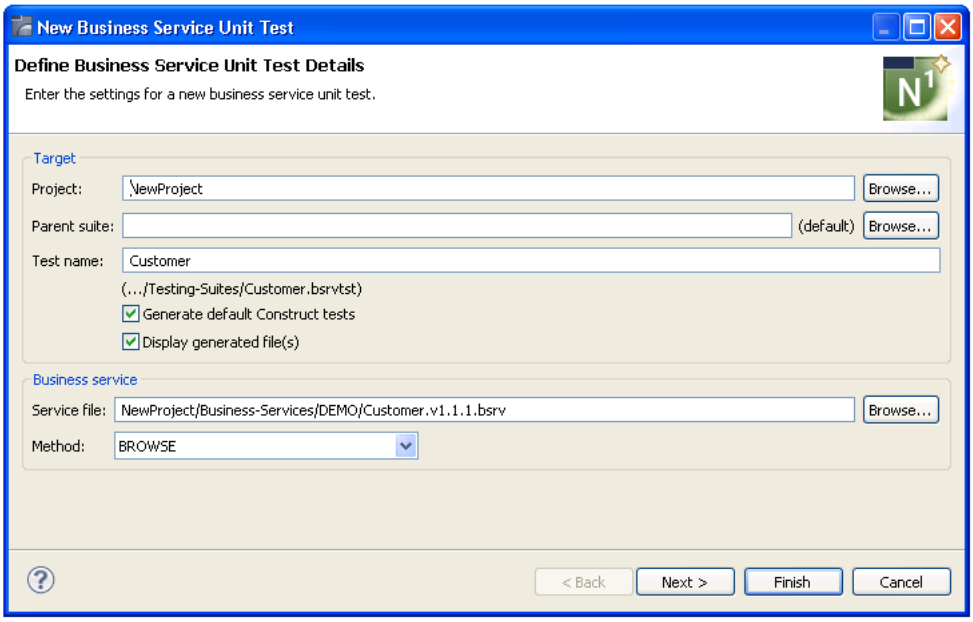
### To generate default unit tests for a business service

1 Select **Testing > Create Unit Test** from the context menu for the business service in the **Navigator** view.

The **Define Business Service Unit Test Details** panel is displayed.

2 Select **Generate default Construct tests**.

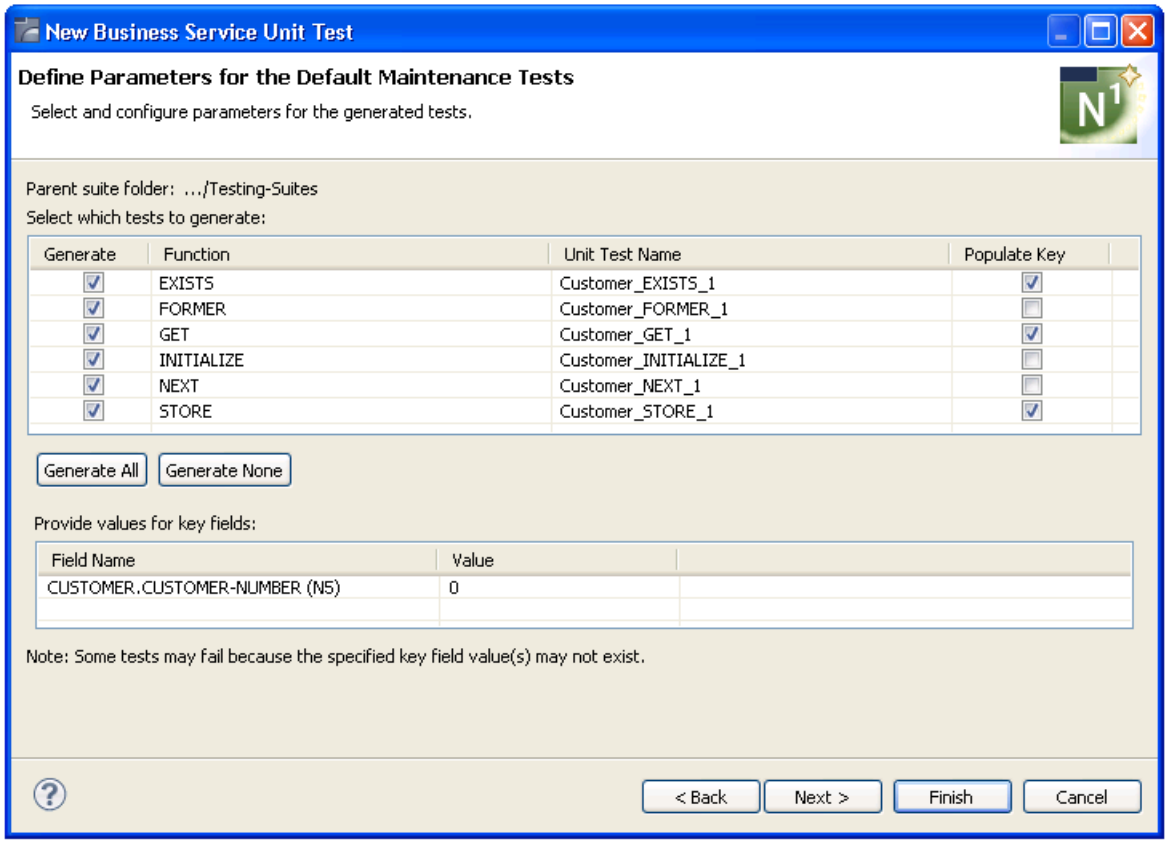
For example:




**Note:** This option is only available when the business service uses one or more subprograms that were generated by an Object-Browse and/or Object-Maint wizard (either Velocity-based or Construct).

3 Select **Next**.

The **Define Parameters for the Default Maintenance Tests** panel is displayed. For example:




**Note:** If the business service does not use any object-maintenance subprograms, the **Define Parameters for the Default Browse Tests** is displayed.

This panel displays the functions defined for all object-maintenance subprograms used by the business service, as well as the key fields. Using this panel, you can:

| Task | Procedure |
|---|---|
| Limit the generation of one or more default tests. | Deselect **Generate** for the unit test(s) you do not want to have generated. To generate unit tests for all functions, select **Generate All**. |
| Limit the generation of all default tests. | Select **Generate None**. |
| Change the default population of key fields. | Select or deselect **Populate Key** for the default unit test(s). When selected, the test for the corresponding function will populate the key field with the value specified in **Value**. |
| Provide a value for a key field. | Select **Value** for the key field and type the value. For example, you can provide a customer number for the Customer number field. |
| Enter details for a date/time field (when defining details for a date or time field). | See *Define Date and Time Details*. |

Default tests can be created for each function defined for the subprogram that does not require an existing record to be on hold. These functions are:

- STORE
- GET
- NEXT
- FORMER
- EXISTS
- INITIALIZE

**Note:** As the DELETE and UPDATE functions require an existing record to be held, default tests are not generated for these functions.

4 Specify a key value in **Value** for each function.

The tests are designed with the assumption that this value exists (i.e., the test will pass when the value exists). The following assumptions are also made:

| Function | Assumption |
|---|---|
| STORE | Assumes the specified key value exists and expects an error from the subprogram saying the value already exists. |
| FORMER | Assumes a key value is not entered and expects a message from the subprogram saying the beginning of file condition has occurred. |
| NEXT | Assumes that the end of file condition has not occurred and expects a message from the subprogram saying the next record was retrieved successfully. |

The key components are those listed in the object PDA for the object-maintenance subprogram as elementary fields under STRUCTURE. For example, MCUSTN, an object-maintenance subprogram used by the Customer business service (located in the SYSBIZDE library), uses the MCUSTA PDA:

```
    1 MCUSTA-ID                          N          5 /* Object identifier
R   1 MCUSTA-ID                                      /* REDEF. BEGIN : MCUSTA-I
    2 STRUCTURE                                      /* To allow MOVE BY NAME
    3 CUSTOMER-NUMBER                    N          5
```

In this example, CUSTOMER-NUMBER will be used as the key.

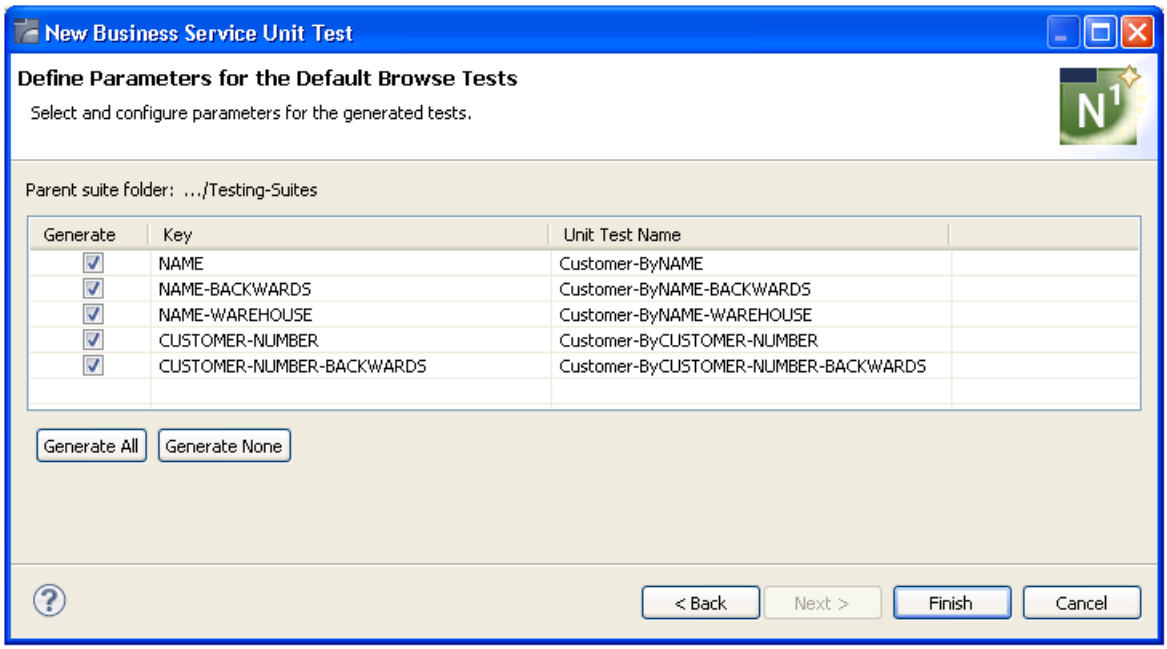
5 Select **Finish**.

Unit tests are created for all available browse keys and any object-maintenance subprogram functions selected on the **Define Parameters for the Default Maintenance Tests** panel.

Or:

Select **Next**.

The **Define Parameters for the Default Browse Tests** panel is displayed. For example:




**Note:** If the business service does not use any object-browse subprograms, **Next** is not available on the **Define Parameters for the Default Maintenance Tests** panel.

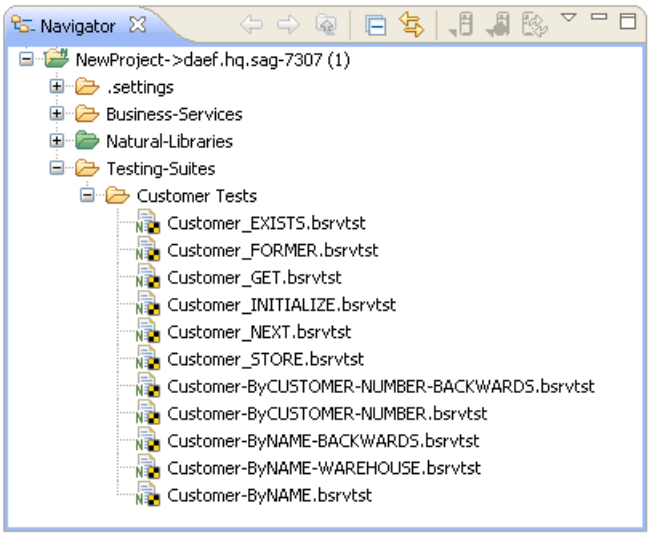
This panel displays the key fields defined for all object-browse subprograms used by the business service. Using this panel, you can:

| Task | Procedure |
|---|---|
| Limit the generation of one or more default tests. | Deselect **Generate** for the unit test(s) you do not want to have generated. To generate unit tests for all keys, select **Generate All**. |
| Change the name of a default unit test. | Type the new name for the unit test on the corresponding line in **Unit Test Name**. |
| Limit the generation of all default tests. | Select **Generate None**. |

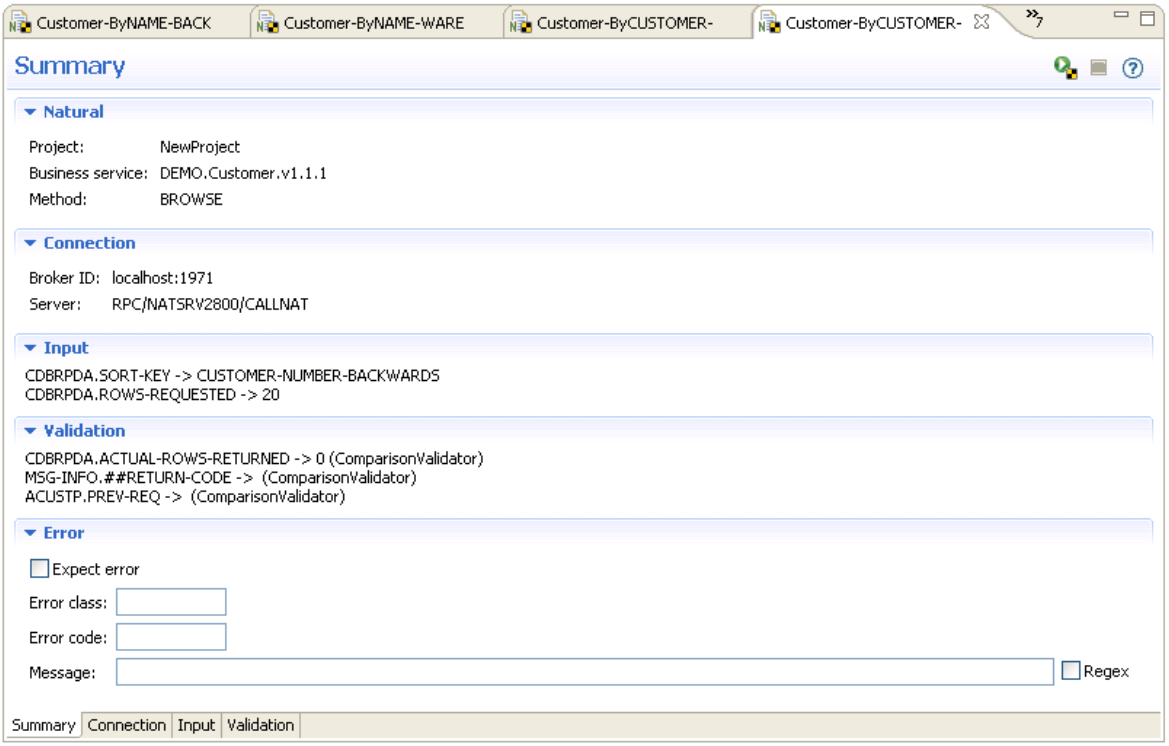
Default tests can be created for each browse key defined for the subprogram. These tests include default validations for items like rows returned and error codes. For a HISTOGRAM key, key value totals can be verified.

6 Select **Finish**.

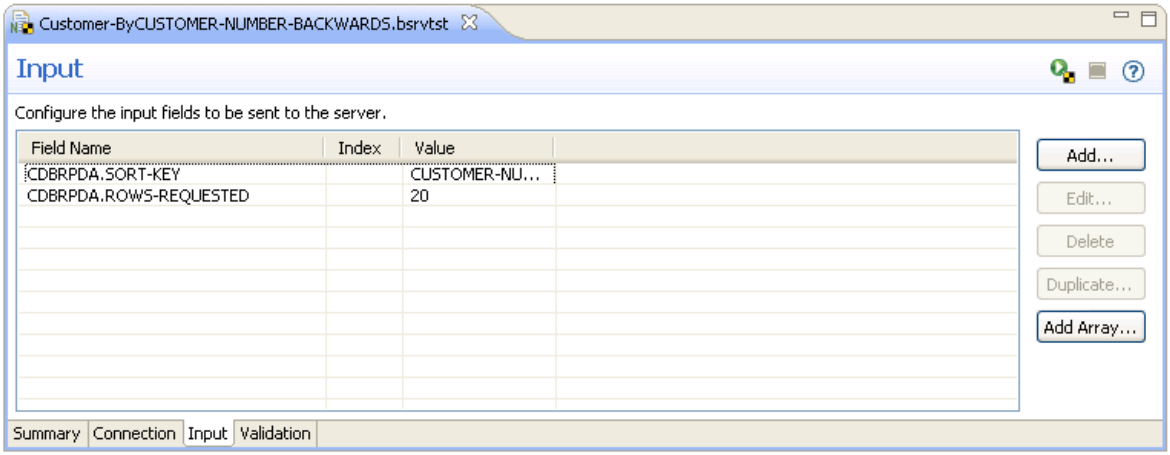
The default unit tests are displayed in the **Testing-Suites** folder in the **Navigator** view. For example:




The tests are also displayed in the editor view. For example:

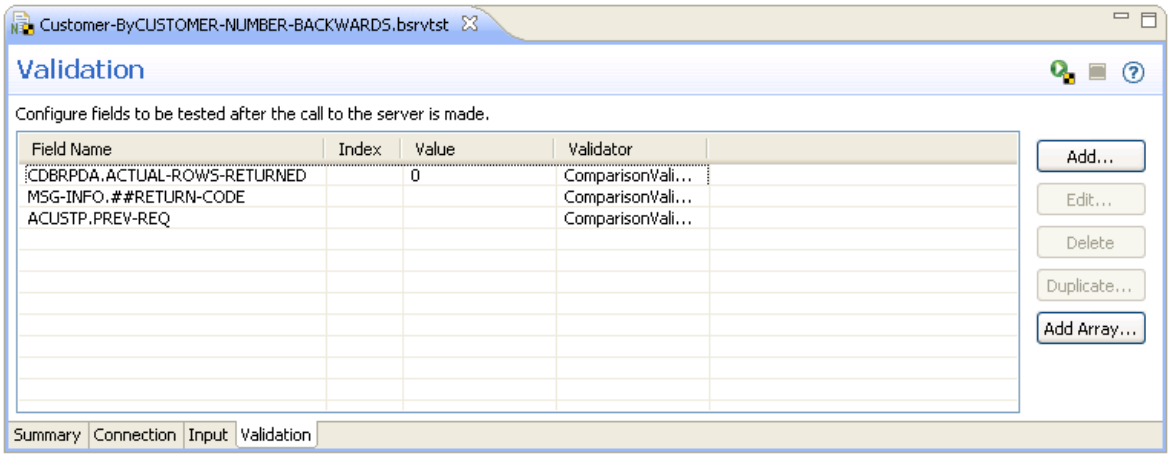



Default input values and validations are created for each unit test. You can change the default values by selecting the appropriate tab. For example, select the **Input** tab to change the input values generated for the test:




**Note:** For more information, see *Configure Input Parameters*.

Select the **Validation** tab to change the validations generated for the test. For example:




**Notes:**

1. For more information, see *Define Validations*.
2. You can create Ant scripts to run unit tests (file extension *.bsrvtst*, *.exttst*, *.nattst*, and *.seqtst*). For information, see *Creating Ant Scripts to Run Unit Tests*.

## Generate Tests for a Natural Subprogram

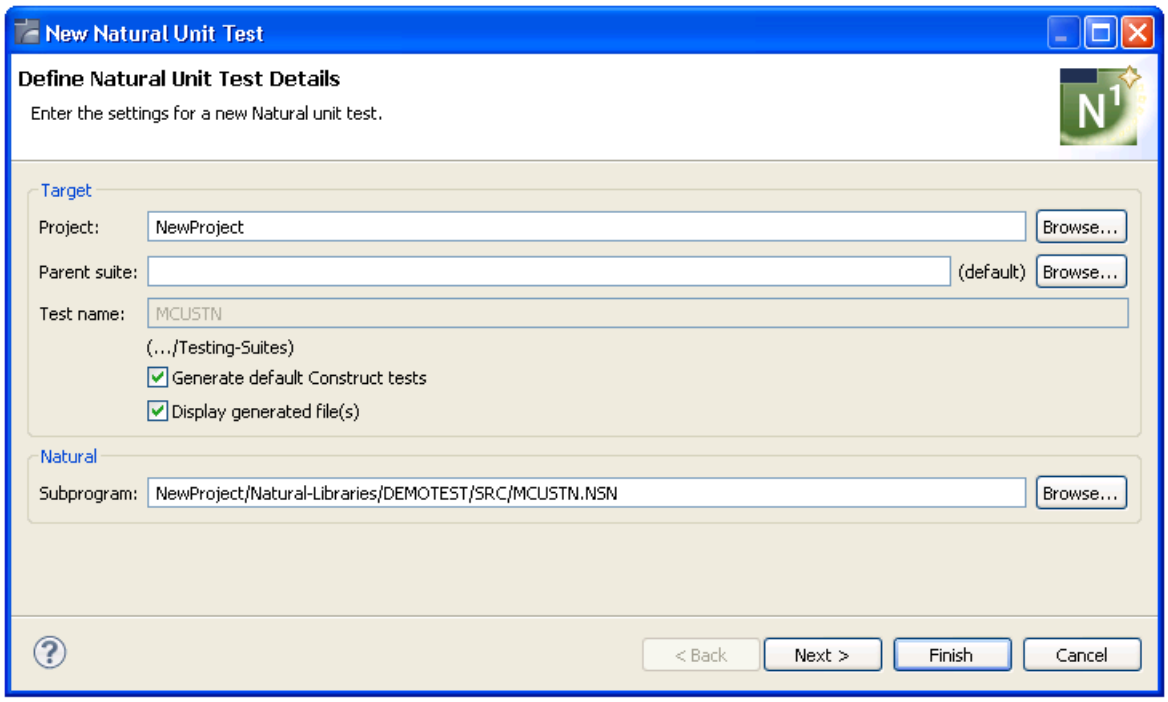
### To generate default unit tests for a Natural subprogram

1 Select **Testing > Create Unit Test** from the context menu for the subprogram in the **Navigator** view.

The **Define Natural Unit Test Details** panel is displayed.

2 Select **Generate default Construct tests**.

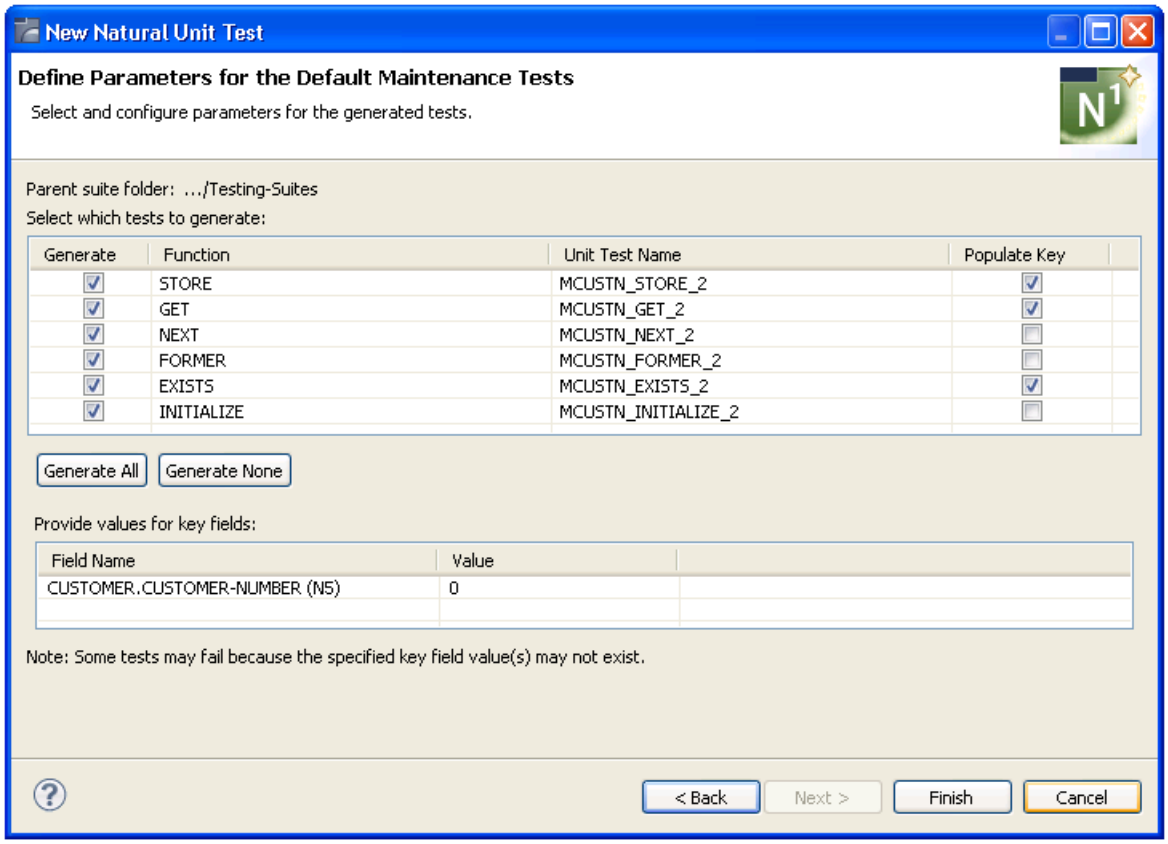
For example:




**Note:** This option is only available when the subprogram was generated by an Object-Browse or Object-Maint wizard (either Velocity-based or Construct).

3 Select **Next**.

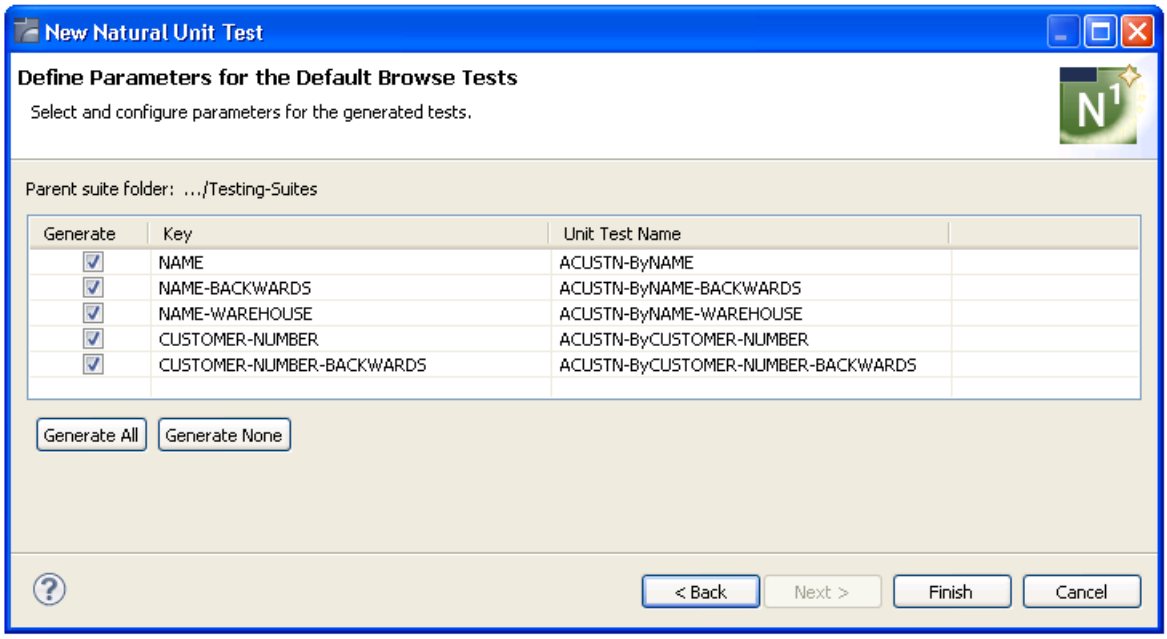
If the subprogram was generated by an Object-Maint wizard, the **Define Parameters for the Default Maintenance Tests** panel is displayed. For example:




This panel is similar to the **Define Parameters for the Default Maintenance Tests** panel for a business service unit test. For a description of this panel, see *Generate Tests for a Business Service*.

Or:

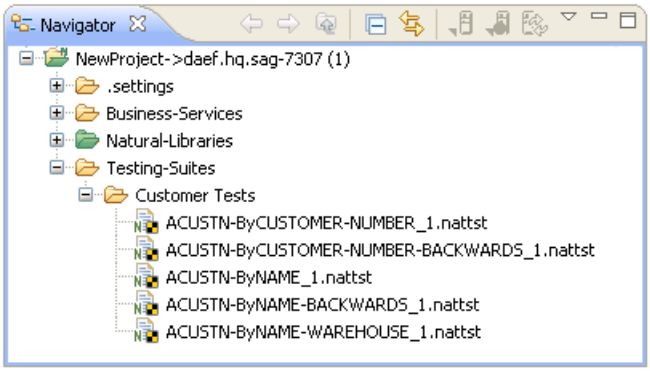
If the subprogram was generated by an Object-Browse wizard, the **Define Parameters for the Default Browse Tests** is displayed. For example:



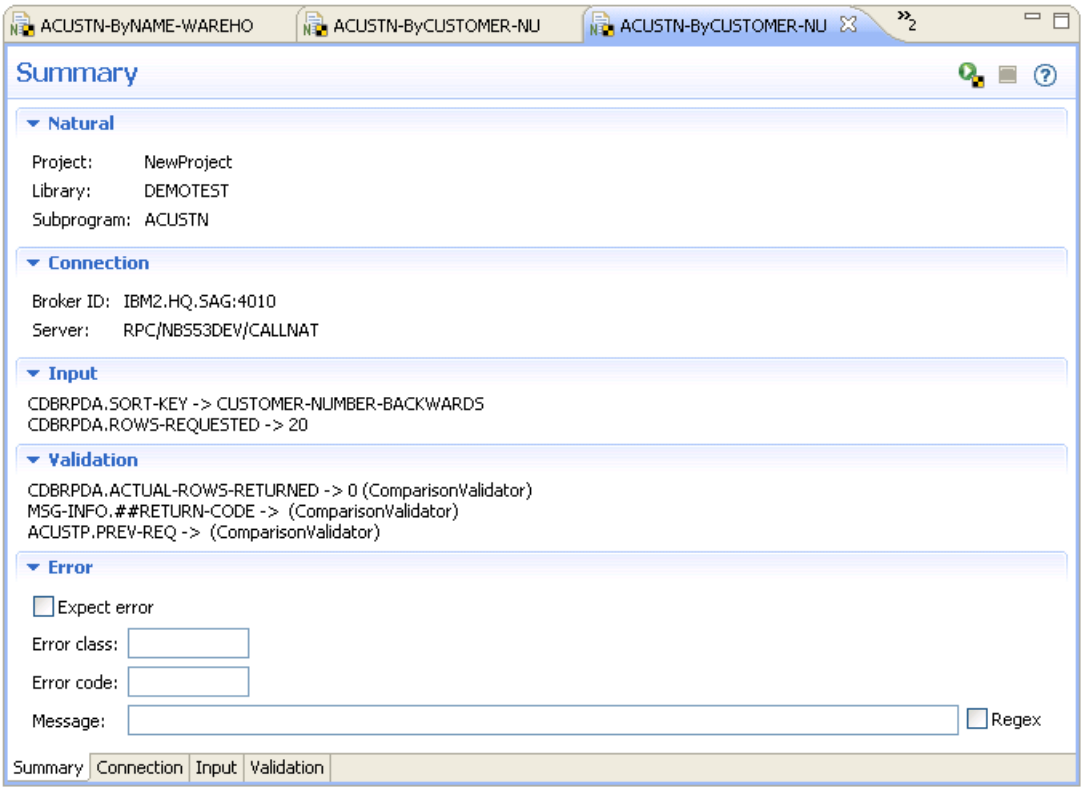

This panel is similar to the **Define Parameters for the Default Browse Tests** panel for a business service unit test. For a description of this panel, see *Generate Tests for a Business Service*.

4 Select **Finish**.

The default unit tests are displayed in the **Testing-Suites** folder in the **Navigator** view. For example:




The tests are also displayed in the editor view. For example:




This editor is similar to the editor for a business service unit test. For a description of the editor, see *Generate Tests for a Business Service*.

## Create a New Unit Test Suite

This section describes how to create a new unit test suite to organize and store your Natural and business service unit tests (file extension *.bsrvtst*, *.exttst*, *.nattst*, and *.seqtst*). The tests are generated into the **Testing-Suites** folder or subfolder within a specified Natural project.

**Note:** Ant scripts for Natural unit tests may contain unit test files existing outside of the **Testing-Suites** folder or subfolder.

### To create a new unit test suite

1 Select **Testing > Create Test Suite** for a project in the **Navigator** view.

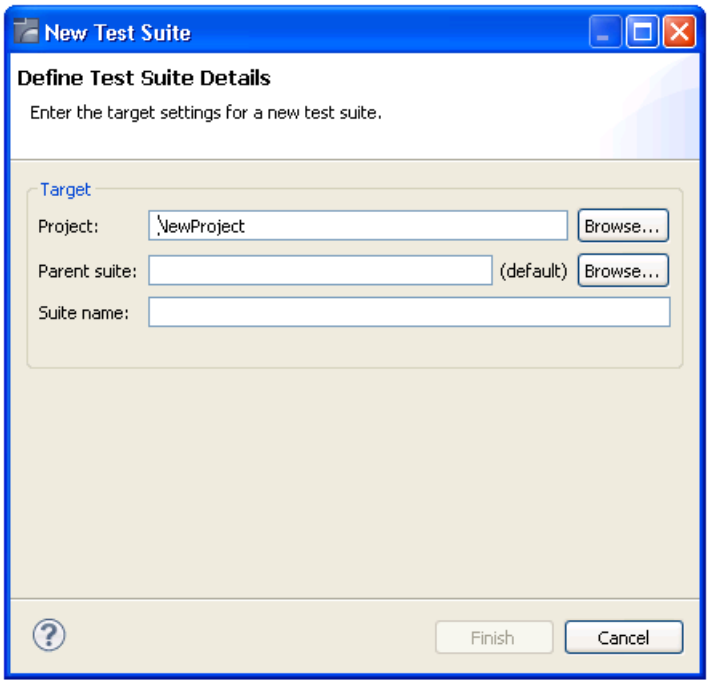
Or:

Select **Testing-Suites > Create Test Suite** in the **Navigator** view.

Or:

Select **Testing-Suites** > *SubfolderName* > **Create Test Suite** in the **Navigator** view.

The **Define Test Suite Details** panel is displayed. For example:




Using this panel, you can:

| Task | Procedure |
|---|---|
| Change the name of the project in which to create the test suite. | Type the name of the Natural project in **Project** or select **Browse** to display a window listing the existing projects for selection.<br><br>**Note:** The project must currently exist. |
| Provide the name(s) of a subfolder(s) in which to save the unit test. If the folder does not currently exist, it will be created for you. | Type the name of the folder in **Parent suite** or select **Browse** to display a window listing the available folders for selection.<br><br>By default, the unit test is stored in the **Testing-Suites** folder in the current project. If you specify a suite folder name, it becomes a subfolder in the **Testing-Suites** folder and the unit test will be stored in that folder. |

2 Type the name of the test suite in **Suite name**.

3 Select **Finish**.

The test suite is generated into the **Testing-Suites** folder or subfolder.

# Create Summary Reports for Unit Test Log Files

This section describes how to create unit test log files and then use the log files to create summary reports. Log files can be created for any subprogram and business service unit test executed within a NaturalONE project.

This section covers the following topics:


- Create Unit Test Log Files
- Generate a Summary Report


## Create Unit Test Log Files

A unit test history log file can be created to save the results of a unit test whenever it is executed (for example, the test name, test status, date/time completed, error messages, etc.). To create these files, you must select the option in the **Preferences** window for **Testing**. For information, see *Set Logging Preferences for Unit Tests*.

## Generate a Summary Report

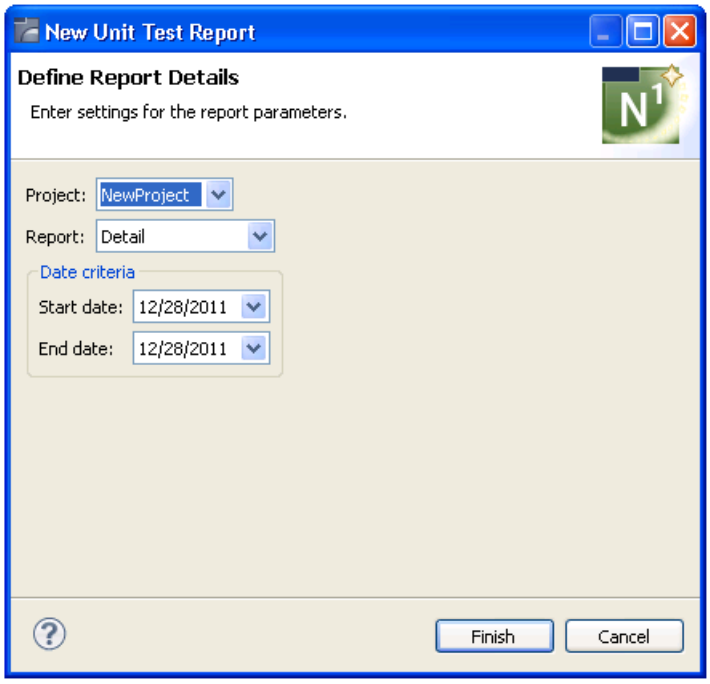
### To generate a report

1 Select **Testing > Create Unit Test Report** for a project in the **Navigator** view.

Or:

Select **Testing-History > Testing > Create Unit Test Report** in the **Navigator** view.

The **Define Report Details** panel is displayed. For example:




**Note:** To change the name of the Natural project, type the name of the project in **Project** or select **Browse** to display a window listing the existing projects for selection.

2 Type or select the name of the report in **Report**.

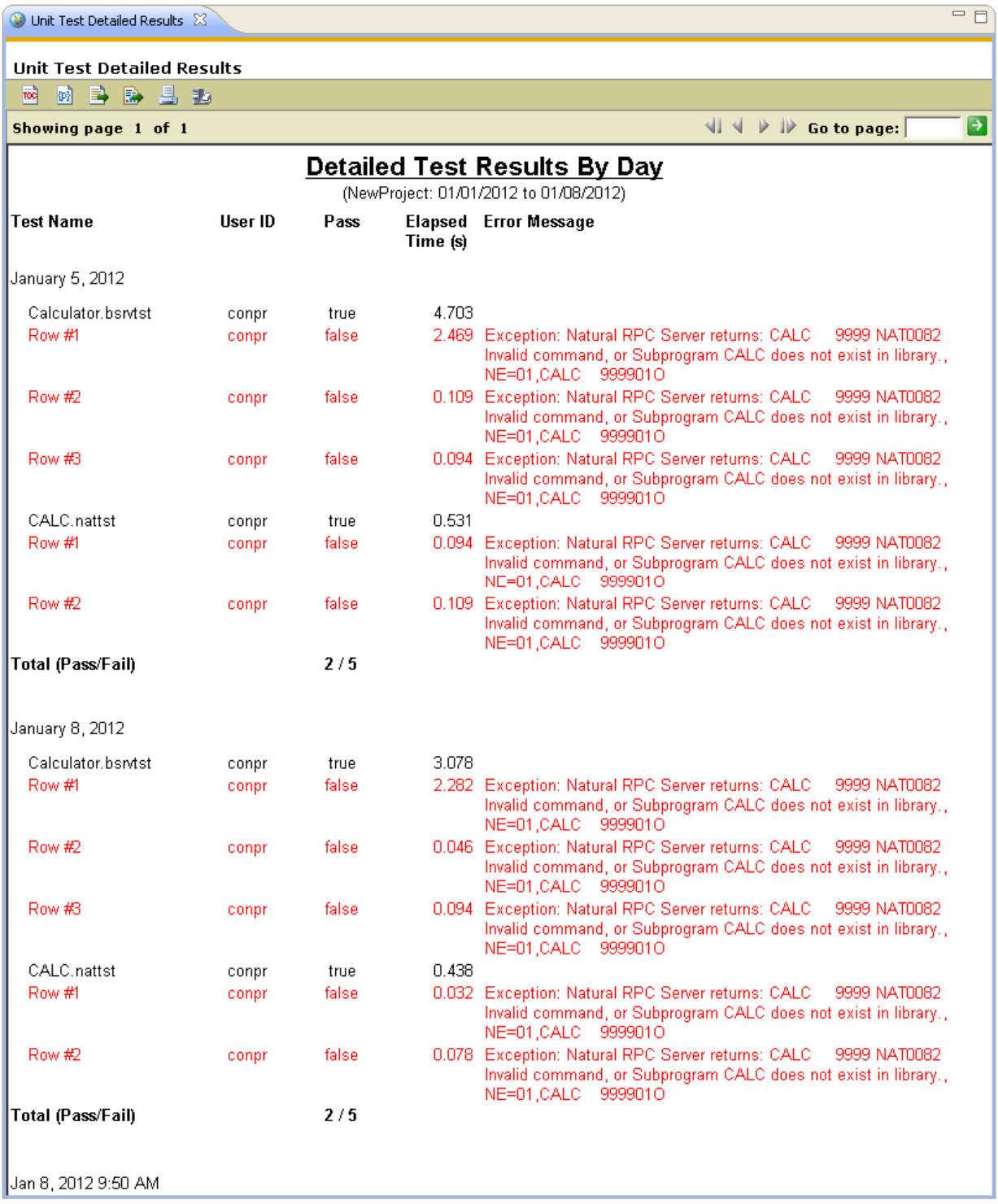
The report types are Detail, Daily summary, History chart and Weekly summary (see below for an example of each report).

3 Select the range of dates for the report in **Date criteria**.

4 Select **Finish**.

The report types are:

**Detail**



Unit Test Detailed Results

**Unit Test Detailed Results**

Showing page 1 of 1 Go to page:

## Detailed Test Results By Day

(NewProject: 01/01/2012 to 01/08/2012)

| Test Name | User ID | Pass | Elapsed Time (s) | Error Message |
|---|---|---|---|---|
| January 5, 2012 | | | | |
| Calculator.bsrvtst | conpr | true | 4.703 | |
| Row #1 | conpr | false | 2.469 | Exception: Natural RPC Server returns: CALC 9999 NAT0082 Invalid command, or Subprogram CALC does not exist in library., NE=01,CALC 9999010 |
| Row #2 | conpr | false | 0.109 | Exception: Natural RPC Server returns: CALC 9999 NAT0082 Invalid command, or Subprogram CALC does not exist in library., NE=01,CALC 9999010 |
| Row #3 | conpr | false | 0.094 | Exception: Natural RPC Server returns: CALC 9999 NAT0082 Invalid command, or Subprogram CALC does not exist in library., NE=01,CALC 9999010 |
| CALC.nattst | conpr | true | 0.531 | |
| Row #1 | conpr | false | 0.094 | Exception: Natural RPC Server returns: CALC 9999 NAT0082 Invalid command, or Subprogram CALC does not exist in library., NE=01,CALC 9999010 |
| Row #2 | conpr | false | 0.109 | Exception: Natural RPC Server returns: CALC 9999 NAT0082 Invalid command, or Subprogram CALC does not exist in library., NE=01,CALC 9999010 |
| **Total (Pass/Fail)** | | **2 / 5** | | |
| January 8, 2012 | | | | |
| Calculator.bsrvtst | conpr | true | 3.078 | |
| Row #1 | conpr | false | 2.282 | Exception: Natural RPC Server returns: CALC 9999 NAT0082 Invalid command, or Subprogram CALC does not exist in library., NE=01,CALC 9999010 |
| Row #2 | conpr | false | 0.046 | Exception: Natural RPC Server returns: CALC 9999 NAT0082 Invalid command, or Subprogram CALC does not exist in library., NE=01,CALC 9999010 |
| Row #3 | conpr | false | 0.094 | Exception: Natural RPC Server returns: CALC 9999 NAT0082 Invalid command, or Subprogram CALC does not exist in library., NE=01,CALC 9999010 |
| CALC.nattst | conpr | true | 0.438 | |
| Row #1 | conpr | false | 0.032 | Exception: Natural RPC Server returns: CALC 9999 NAT0082 Invalid command, or Subprogram CALC does not exist in library., NE=01,CALC 9999010 |
| Row #2 | conpr | false | 0.078 | Exception: Natural RPC Server returns: CALC 9999 NAT0082 Invalid command, or Subprogram CALC does not exist in library., NE=01,CALC 9999010 |
| **Total (Pass/Fail)** | | **2 / 5** | | |

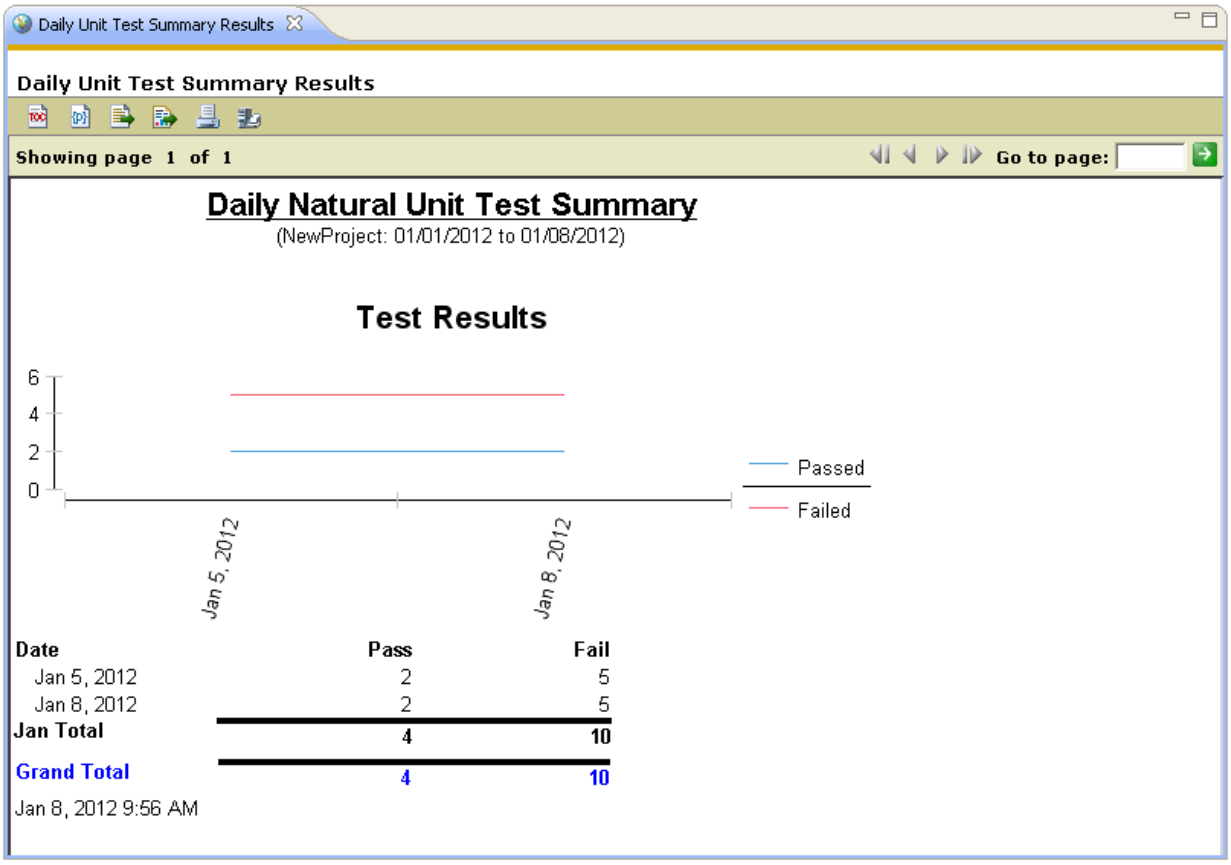
Jan 8, 2012 9:50 AM

This report shows:

- Name project containing the tests, as well as the range of dates included in the report
- Name of each test

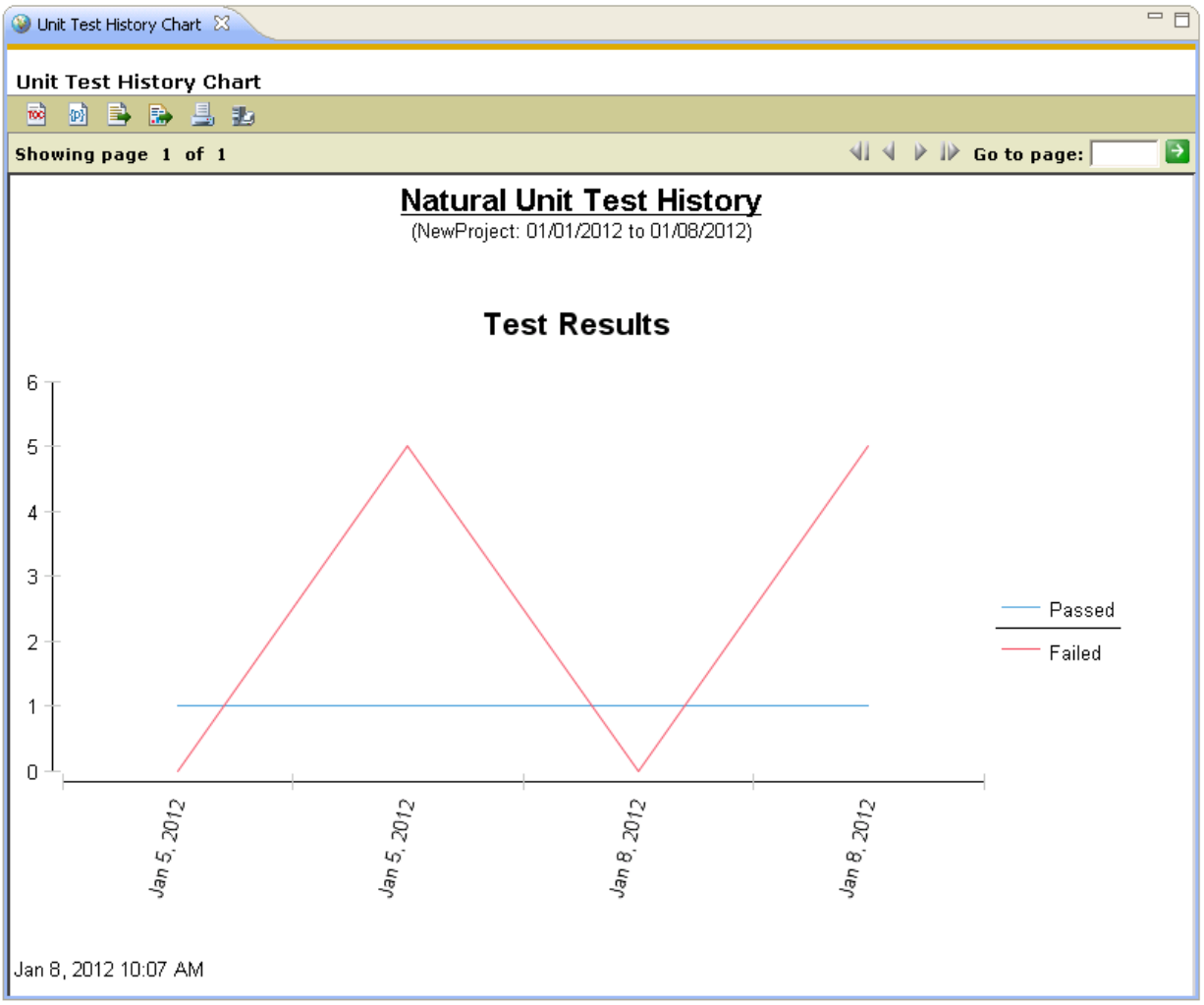
- User ID of the person who executed the unit test (or Unknown if the user ID cannot be determined)
- Whether the test passed (true) or failed (false)
- Elapsed time (in seconds) that the test took to run
- Error message for tests that failed
- Total number of tests that passed or failed
- Date and time the report was created

### Daily Summary



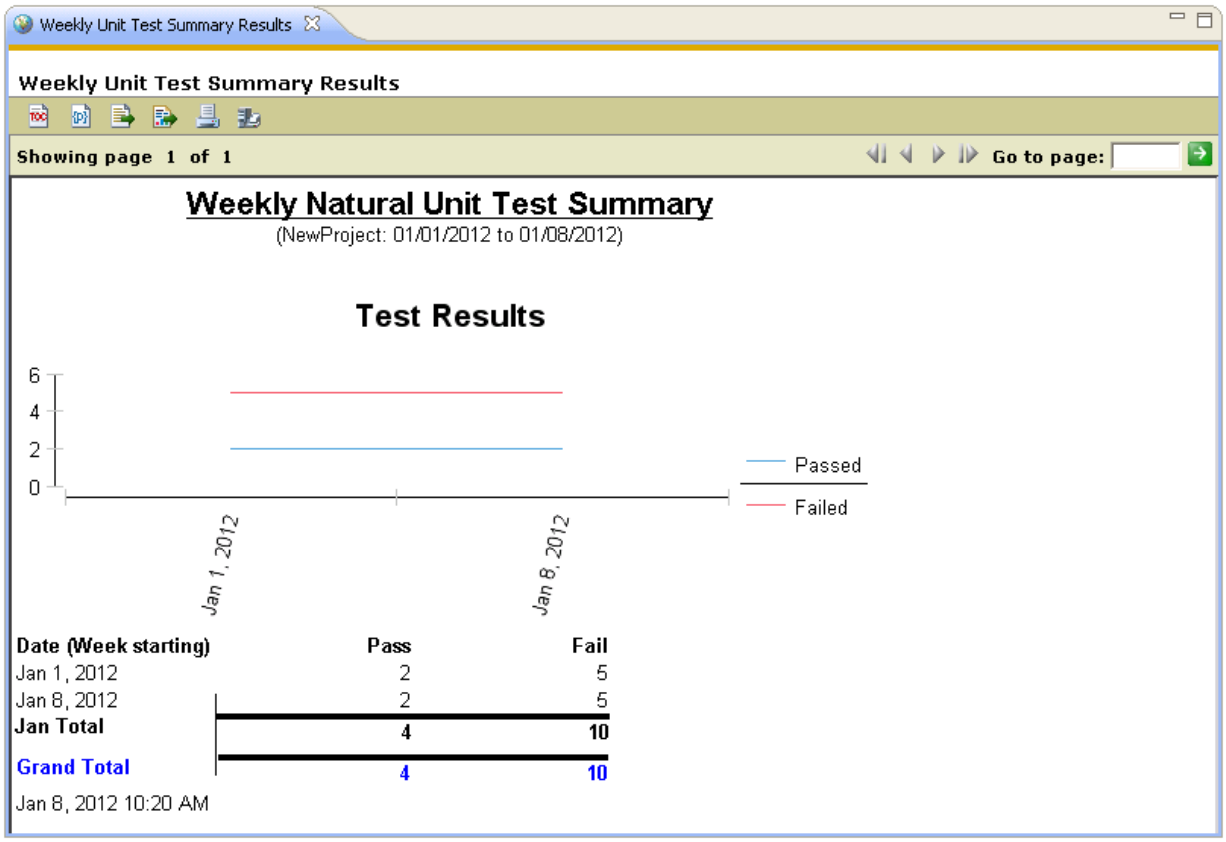

This report calculates and displays a daily Pass/Fail summary from every unit test executed within a selected range of dates.

### History Chart




This report provides a graph of the Pass/Fail count for each **Testing-History** log file created within a selected range of dates.

**Weekly Summary**




This report calculates and displays a weekly Pass/Fail summary from every unit test executed within a selected range of dates.

## Use the Dependencies View

When a generated module is open in the editor view, the **Dependencies** view displays dependencies between business service and Natural unit tests and the business services and Natural subprograms they execute. This section describes the nodes contributed to the view for these resources. The following topics are covered:

- Business Service Unit Test Resources
- Natural Subprogram Unit Test Resources

**Notes:**

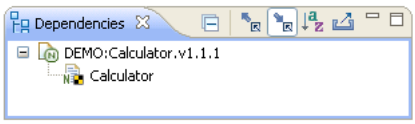
1. Select to sort the resources alphabetically.
2. Select to export a textual representation of the visible nodes in the view to a file.
3. When a supporting resource cannot be found locally using the project steplib chain and project references, "<Unknown>" is displayed with the name of the resource. If the unknown module(s)

is not shipped with the Construct runtime project, either manually download it from the server or create it locally. If the module(s) is shipped with the Construct runtime project, add the project. For information, see NaturalONE's *Code Generation* documentation.

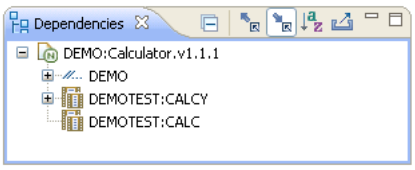
4. For more information about the **Dependencies** view, see the description of the source editor in *Using NaturalONE*.

### Business Service Unit Test Resources

When a business service unit test is open in the editor view, the root node displays the name of the business service unit test. In caller mode (), no child nodes are displayed because no other **Dependencies** view objects depend on this business service unit test file. For example:

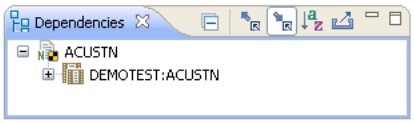



In callee mode (), the child nodes display the name of the business service that the unit test executes, along with the names of the supporting business service resources and the names of the libraries and projects in which they are located. For example:




### Natural Subprogram Unit Test Resources

When a Natural subprogram unit test is open in the editor view, the root node displays the name of the unit test. In caller mode (), no child nodes are displayed because no other **Dependencies** view objects depend on a unit test file; in callee mode (), the child node displays the name of the Natural subprogram that the unit test executes, along with the names of the supporting Natural resources and the names of the libraries and projects in which they are located. For example:



# 7 Create an External Data Unit Test


- Create the Unit Test ..... 86
- Configure Column Mappings and Sample Data ..... 91

This section describes how to create a unit test that accepts input and/or validations from a CSV file (file extension *.csv*). You can create a unit test once and then provide a data file containing different input or validations to run iterations of the test. The wizard creates a unit test file that accepts data from the CSV file.

**Note:** Similar to other unit tests, external data unit tests can be run from the unit test Ant script. For information, see *Creating Ant Scripts to Run Unit Tests*.

## Create the Unit Test

### To create an external data unit test

1 Select **Testing > Create External Data Unit Test** for a project in the **Navigator** view.

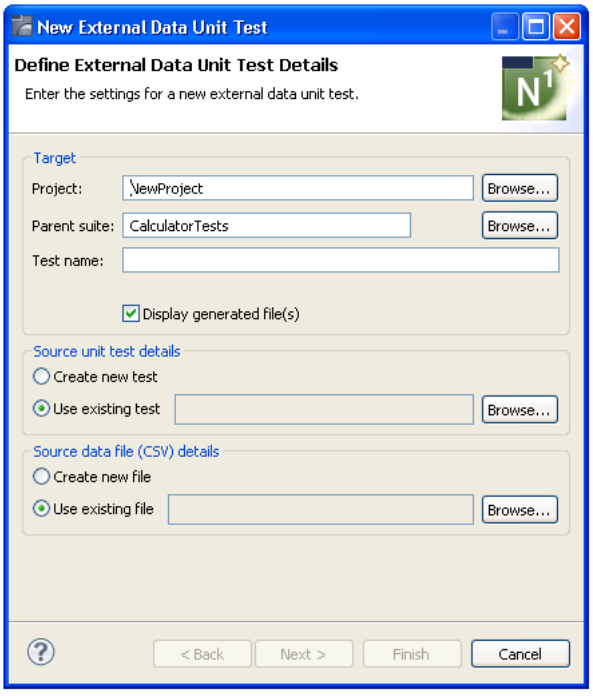
Or:

Select **Testing-Suites > Create External Data Unit Test** in the **Navigator** view.

Or:

Select **Testing-Suites>** *SubfolderName* **> Create External Data Unit Test** in the **Navigator** view.

The **Define External Data Unit Test Details** panel is displayed. For example:



Using this panel, you can:

| Task | Procedure |
|---|---|
| Change the name of the project in which to create the external data unit test. | Type the name of the Natural project in **Project** or select **Browse** to display a window listing the existing projects for selection.<br><br>**Note:** The project must currently exist. |
| Provide the name(s) of a subfolder(s) in which to save the external data unit test. If the folder does not currently exist, it will be created for you. | Type the name of the folder in **Parent suite** or select **Browse** to display a window listing the available folders for selection.<br><br>By default, the unit test is stored in the **Testing-Suites** folder in the current project. If you specify a suite folder name, it becomes a subfolder in the **Testing-Suites** folder and the unit test will be stored in that folder. |

2 Type the name of the external data unit test in **Test name**.

3 Select an existing business service or Natural unit test in the **Source unit test details** section.

The selected unit test will be executed for each row in the data file. To display the available unit test files for selection, select **Browse** for **Use existing test**. Optionally you can create a new business service or Natural unit test. For information, see *Create a New Unit Test*.

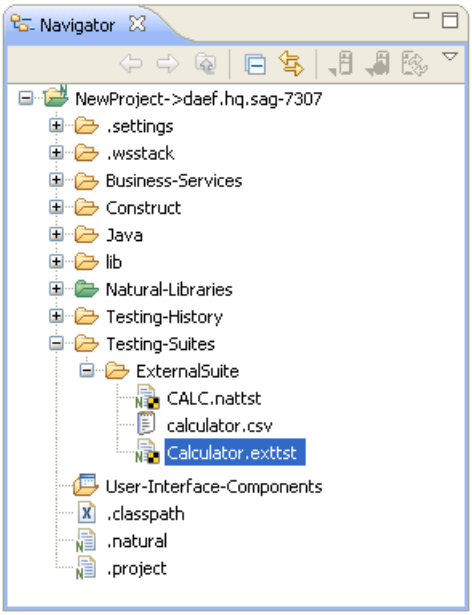
4 Select an existing data file in the **Source data file (CSV) details** section.

To display the available CSV data files for selection, select **Browse** for **Use existing file**. Optionally you can create a new data file. For information, see *Create a New Data File*.

**Note:** A wizard is available to record the sample data used to test a business service or subprogram directly and then export the data to a CSV file. For information, see *Export Test Data to a CSV File*.

5 Select **Finish**.

The external data unit test file is generated into the **Testing-Suites** folder (or subfolder) and listed in the **Navigator** view. For example:




The *.exttst* file is also displayed in the editor view.

**Note:** The *.csv* file and/or the *.nattst*/*.bsrvtst* files may also be created.

6 Define the configuration settings for the unit test in the editor view.

For information, see *Configure Column Mappings and Sample Data*.

7 Select the **Connection** tab and define the connection settings for the unit test.

For information, see *Define Connections*.

8 Save the settings.

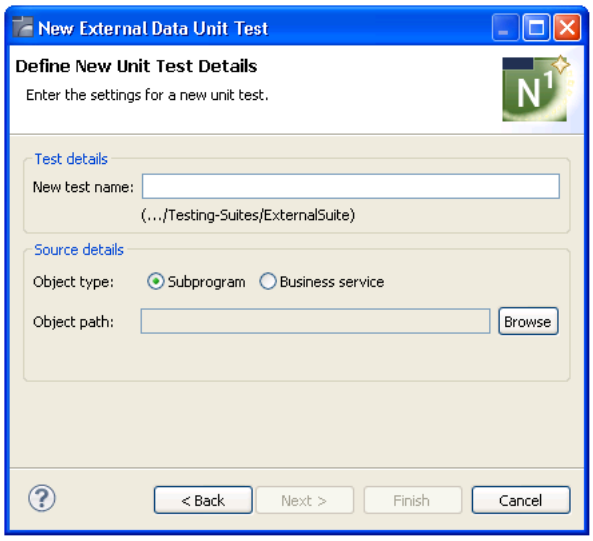
## Create a New Unit Test

### To create a new unit test

1 Select **Create new test** in the **Source details** section on the **Define External Data Unit Test Details** panel.

2 Select **Next**.

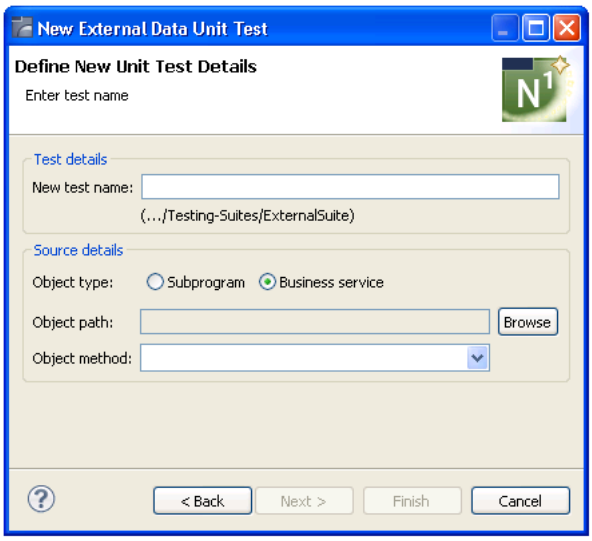
The **Define New Unit Test Details** panel is displayed. For example:




3 Type the name of the unit test in **New test name**.

4 Select the object type for the source unit test in **Object type**.

You can select either **Subprogram** (the default) or **Business service**. When **Business service** is selected, an additional field is added to the panel. For example:




5 Select **Browse** in **Object path**.

A list of available business service or subprogram unit test files is displayed. Select the unit test to use for the external data unit test and select **OK**.

6 For a business service unit test, select the method to test in **Object method**.

7 Select **Finish** to create the external data unit test and new unit test.

Or:

Select **Next** to create a new data file.

**Note:** This option is only available when **Create new file** is selected on the **Define External Data Unit Test Details** panel.

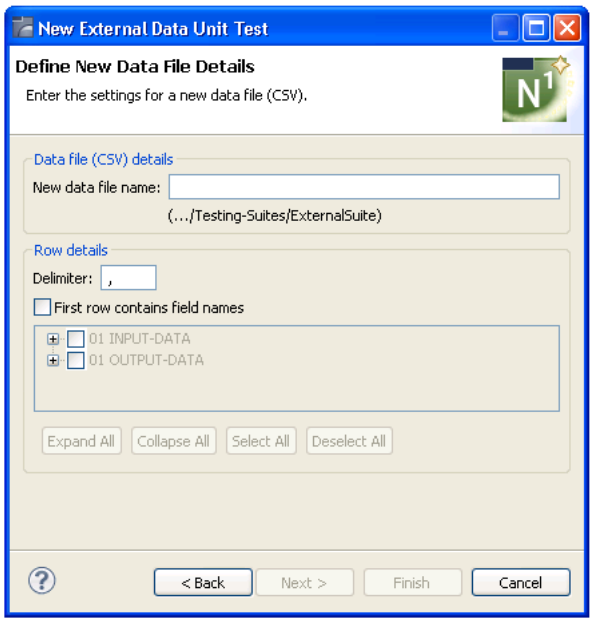
## Create a New Data File

### To create a new data file

1 Select **Create new file** in the **Source data file (CSV) details** section on the **Define External Data Unit Test Details** panel.

2 Select **Next**.

The **Define New Data File Details** panel is displayed. For example:




**Note:** If **Create new test** on the **Define External Data Unit Test Details** panel is also selected, the **Define New Unit Test Details** panel is displayed before this panel.

3 Type the name of the data file in **New data file name**.

Using this panel, you can:

| Task | Procedure |
|---|---|
| Change the character used to separate entries in the first row of the CSV file. | Type a new character in **Delimiter**. |
| Reserve the first row in the CSV file for the field names. | Select **First row contains field names**. At runtime, the first row in the CSV file is reserved for field names.<br><br>**Note:** When selecting fields for the first row in a CSV file, you cannot specify the number of occurrences of an array to include. By default, a maximum of five occurrences of each array will be included. To add and/or remove occurrences from the generated CSV file, you must edit the file manually. |
| Display fields that can be selected for the first row of the CSV file. | Select **Expand All**. To close the tree view, select **Collapse All**. |
| Select fields to be included in the first row of the CSV file. | Select **Select All** and then deselect the fields you do not want to include in the CSV file. To deselect all fields, select **Deselect All**. |

4 Select **Finish** to create the external data unit test, a new data file, and optionally, a new unit test.

## Configure Column Mappings and Sample Data

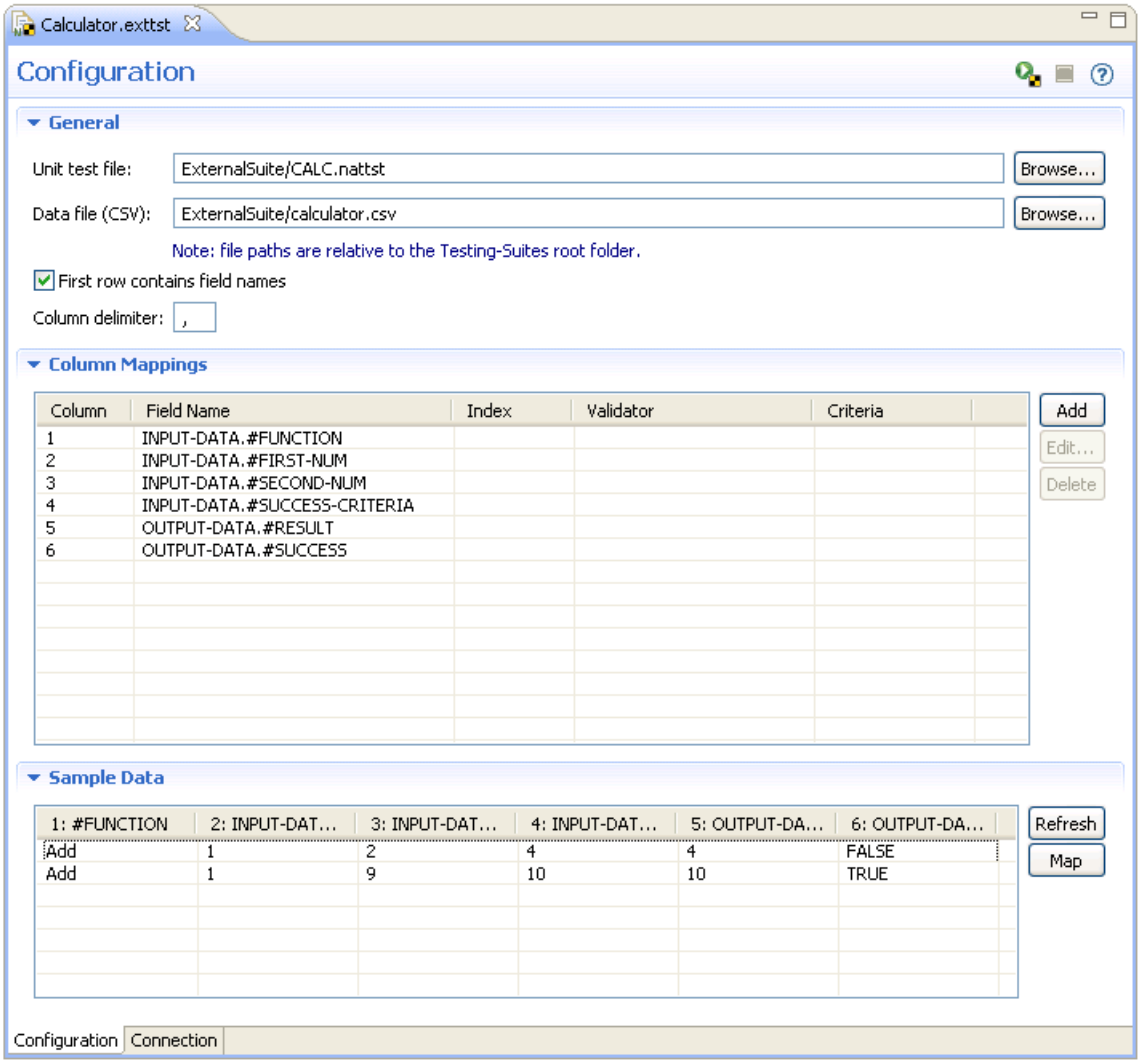
This section describes how to map columns in the CSV file (file extension *.csv*) to fields in the PDA used by the business service or subprogram unit test. The following CSV file was used for examples:

```
#FUNCTION,INPUT-DATA.#FIRST-NUM,INPUT-DATA.#SECOND-NUM,INPUT-DATA.#SUCCESS-CRITERIA,OUTPUT-DATA.#RESULT,OUTPUT-DATA.#SUCCESS
Add,1,2,3,3,FALSE
Add,1,9,10,10,TRUE
```

### To configure column mappings and sample data

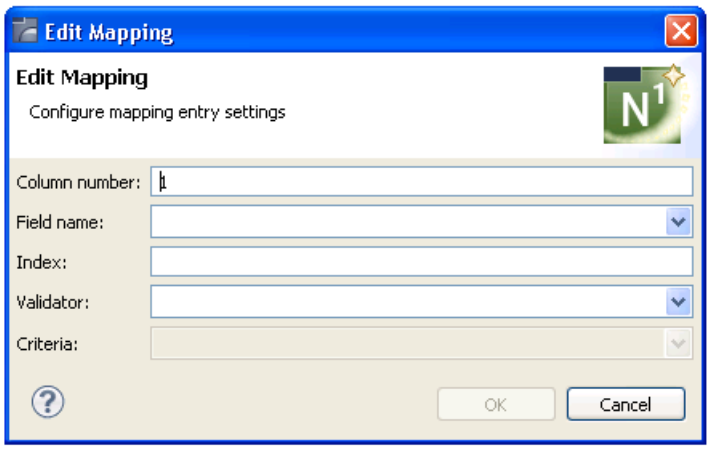
1 Select the **Configuration** tab in the editor for the external data unit test.

For example:




2 Select **Add** in the **Column Mappings** section.

The **Edit Mapping** window is displayed. For example:




The number of the first unmapped column is displayed in **Column number**. You can change this number to define the mapping for another column.

3 Select the name of the field to use for this column in **Field name**.

4 Type the index position in **Index** (used when the field is an array).

5 Select the type of validator to use for the field in **Validator**.

The type of validator to use depends on the type of data in the field. The available validators are:

- BooleanValidator
- ByteValidator
- ComparisonValidator (displays a combo box with the options: ">", "<", "=", "<=", ">=")
- DateValidator
- DecimalValidator
- IntegerValidator
- RegexValidator (creates regular expressions to validate the contents of a field)
- StringValidator
- TimeValidator

6 Select **OK**.

The new column mapping is added to the list of mappings on the **Configuration** tab.

7 Continue adding column mappings until all columns used for the test have been added.

- To revise a mapping, select the mapping in **Column Mappings** and select **Edit**. The **Edit Mapping** window is displayed to change the mapping.

- To remove a mapping, select the mapping in **Column Mappings** and select **Delete**. The mapping is removed from **Column Mappings**.

Optionally, you can use the **Configuration** tab to:

| Task | Procedure |
|---|---|
| Change the name and/or location of the unit test file used for the external data unit test. | Type the name of the unit test in **Unit test file** or select **Browse** to display a window listing the existing unit test files for selection.<br>**Note:** The unit test must currently exist. |
| Change the name and/or location of the CSV file containing field names and input for the external data unit test. | Type the name of the CSV file in **Data file** or select **Browse** to display a window listing the existing CSV files for selection.<br>**Note:** The CSV file must currently exist. |
| Reserve the first row in the CSV file for the field names. | Select **First row contains field names**. At runtime, the first row in the CSV file is reserved for field names.<br>**Note:** When selecting fields for the first row in a CSV file, you cannot specify the number of occurrences of an array to include. By default, a maximum of five occurrences of each array will be included. To add and/or remove occurrences from the generated CSV file, you must edit the file manually. |
| Change the delimiter character used to separate columns in the CSV file. | Type a new delimiter character in **Column delimiter**. |
| Retrieve sample data from the CSV file. | Select **Refresh** in the **Sample Data** section. The first 20 rows in the CSV file are retrieved.<br>**Tip:** To apply changes to the external data file to the unit test, use this option with the **Map** option. |
| Map new sample data to the columns. | Select **Map** (enabled when the **First row contains field names** option is selected). A confirmation window is displayed, indicating that all current column mappings will be removed. Select **Yes** to delete the old mappings and apply the new mappings. |

8 Save the configuration settings.

# 8 Create a Sequence Unit Test


- Create the Unit Test ..... 97
- Use the Sequence Unit Test Editor ..... 100
- Use the Dependencies View ..... 109

This section describes how to create a sequence unit test (file extension *.seqtst*), a type of unit test that executes a sequence of test steps in a specified order. Each test step executes a business service or Natural unit test and, optionally, copies data between steps, applies field overrides, defines validation overrides. and/or applies method overrides (business service unit tests only). These overrides do not physically change the existing unit test files; the values are only changed in memory prior to execution of the files.

For example, a sequence test can have the following two steps:

1. Invoke a unit test for a Construct-generated object-maintenance subprogram and attempt to retrieve (GET) a data record.
2. Re-invoke the same test, but apply a field override that attempts to update the record. In addition, copy all data from Step 1 and pre-configure each input field.

There are several methods you can use to create a sequence unit test, depending on your requirements. These methods include:

- Create one generic business service or Natural unit test and then create a sequence unit test containing several test steps that reference the same generic unit test, but use a different field override.

  For example, you can create a generic Natural unit test called WAREHOUSE.nattst and then create a unit test that reference a sequence of unit tests to override the value of WAREHOUSE.#FUNCTION, such as WAREHOUSE_GET.nattst, WAREHOUSE_NEXT.nattst, etc.
- Create several business service and/or Natural unit tests that reference the same subprogram/PDA and then create a sequence unit test that references each unit test in a specified sequence.

  For example, you can create a unit test for each warehouse function, such as WAREHOUSE_GET.nattst, WAREHOUSE_NEXT.nattst, etc., and then create a unit test that invokes these tests in a specified sequence.
- Create several business service and/or Natural unit tests that reference different subprograms/PDAs and then create a sequence unit test that references each unit test in a specified sequence and copies data from one test to the next.
- Create a sequence unit test and one or more unit tests to use for the test.

## Create the Unit Test

This section describes how to use the wizard to create a sequence unit test.

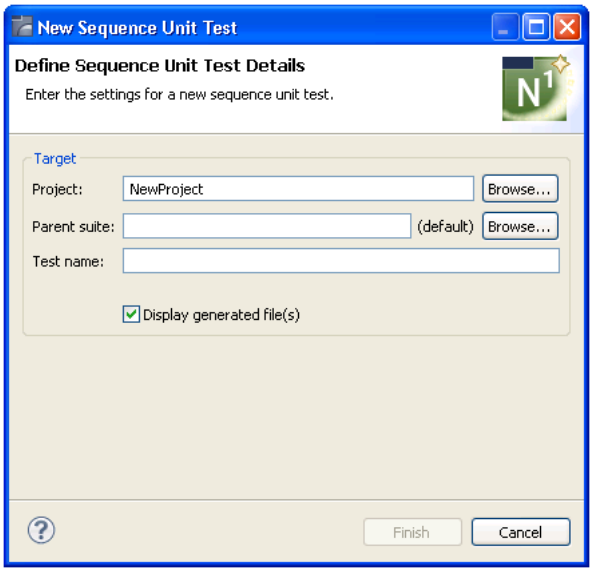
### To create a sequence unit test

1 Open the context menu for one of the following items in the **Navigator** view:

- Project folder
- **Testing-Suites** folder or subfolder
- One or more business service and/or Natural unit test files (using standard selection techniques). The tests can reference the same subprogram/PDA or different subprograms/PDAs. The wizard will create one test step in the generated sequence unit test for each unit test selected in the **Navigator** view.

2 Select **Testing > Create Sequence Unit Test**.

The **Define Sequence Unit Test Details** panel is displayed. For example:



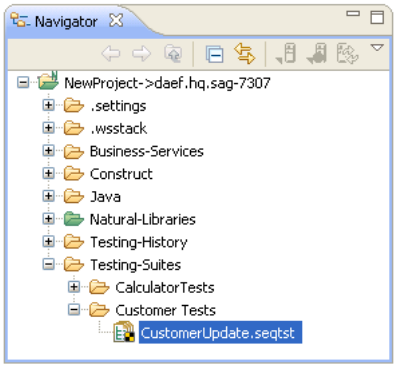

3 Type the name of the sequence unit test in **Test name**.

Optionally, you can:

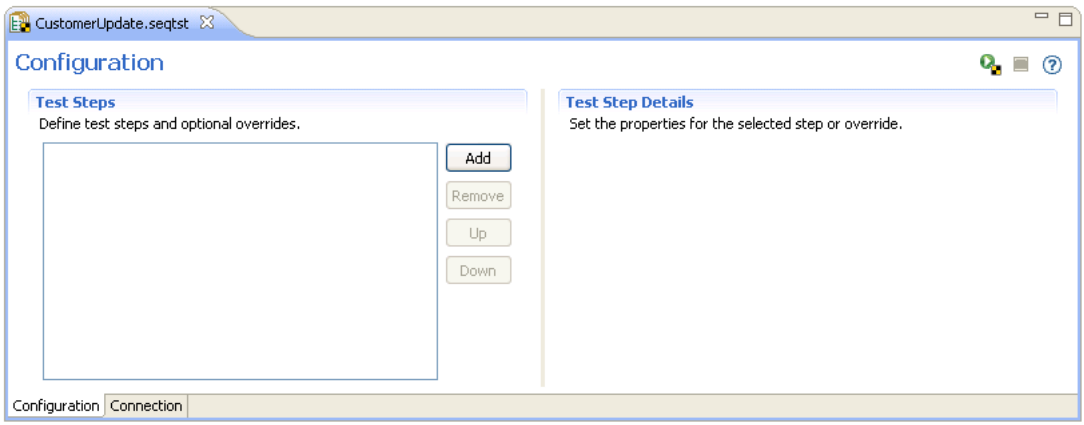
| Task | Procedure |
|---|---|
| Change the name of the project in which to create the sequence unit test. | Type the name of the Natural project in **Project** or select **Browse** to display a window listing the existing projects for selection.<br><br>**Note:** The project must currently exist. |
| Provide the name(s) of a subfolder(s) in which to save the sequence unit test. If the folder does not currently exist, it will be generated for you. | Type the name of the folder in **Parent suite** or select **Browse** to display a window listing the available folders for selection.<br><br>By default, the unit test is stored in the **Testing-Suites** folder in the current project. If you specify a suite folder name, it becomes a subfolder in the **Testing-Suites** folder and the unit test will be stored in that folder. |

4 Select **Finish**.

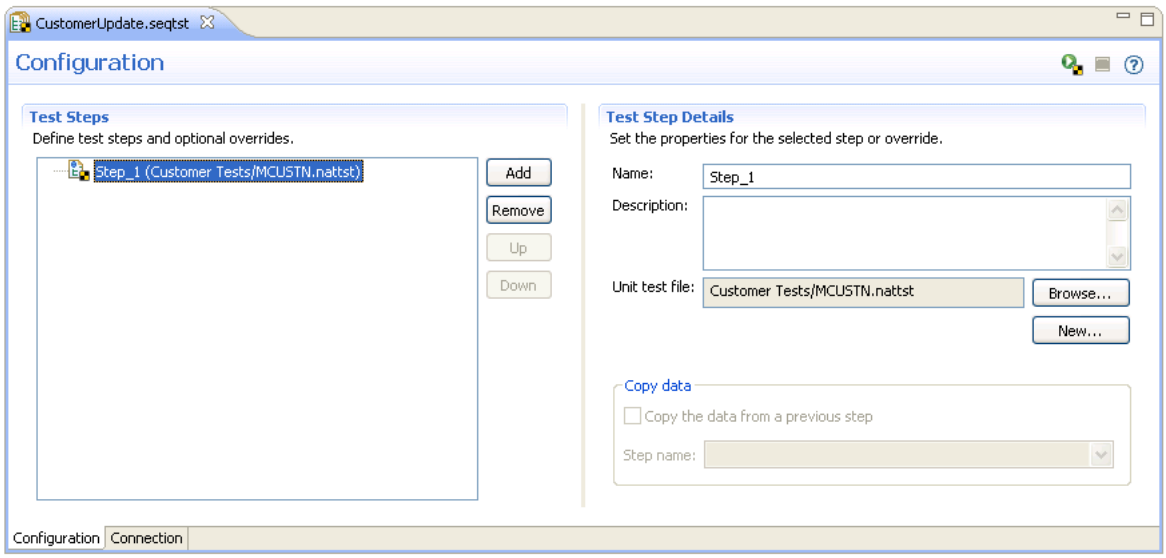
The sequence unit test file is generated into the **Testing-Suites** folder (or subfolder) and listed in the **Navigator** view. For example:



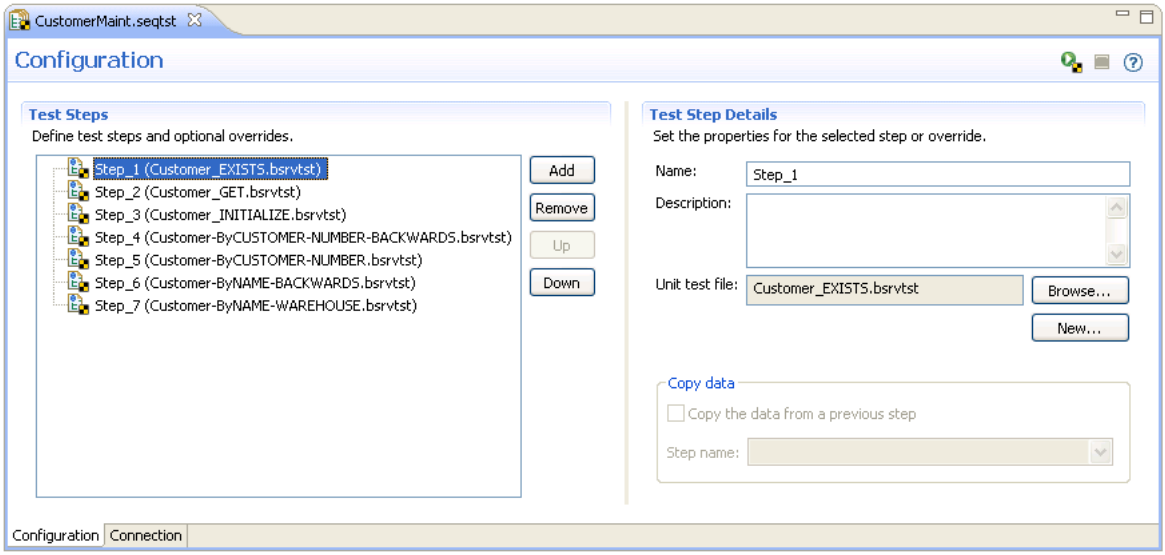

The *.seqtst* file is also displayed in the editor view. For example:



If one unit test file was selected in the **Navigator** view, a default test step is created for that file. For example:




If several unit test files were selected in the **Navigator** view, one test step is created for each test. For example:



## Use the Sequence Unit Test Editor

This section describes how to use the sequence unit test editor. The following topics are covered:

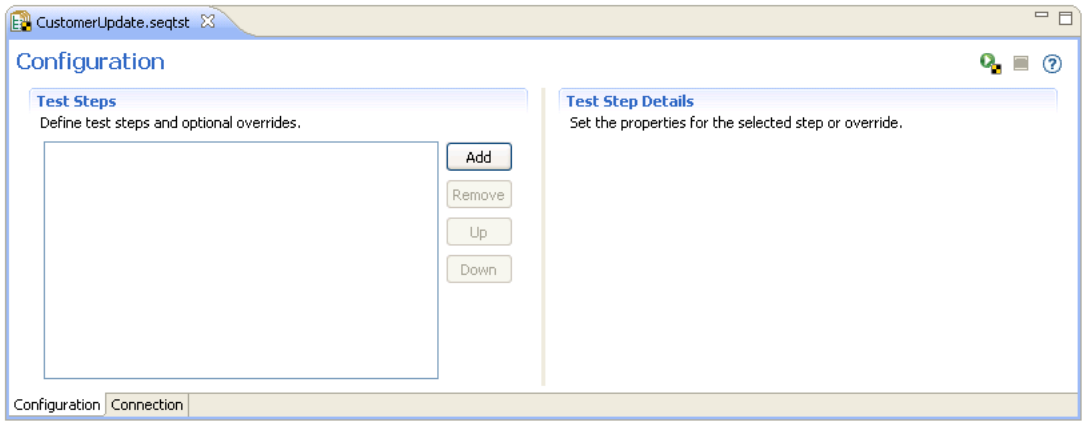
- Add Test Steps
- Copy Data from a Previous Step
- Add an Input Override
- Add a Validation Override
- Add a Method Override

**Notes:**

1. For information about the **Connections** tab, see *Define Connections*.
2. For general information about using the test editors, see *Features of the Test Editors*

### Add Test Steps

This section describes how to add test steps to execute business service and/or Natural unit tests in a specified order. Each test step executes one existing unit test and, optionally, copies data between steps, applies field overrides, and/or defines validation overrides. In the following example, the sequence unit test is generated from the context menu for a project and no steps are created. For example:

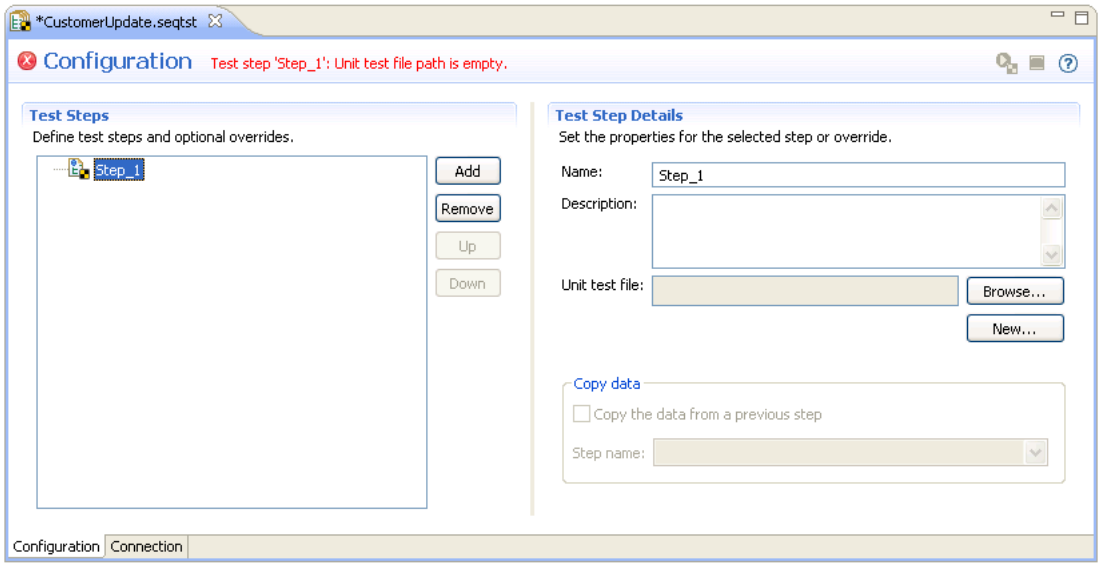



**Note:** To resize the editor sections, select the sash and move it left or right.

**To add test steps**

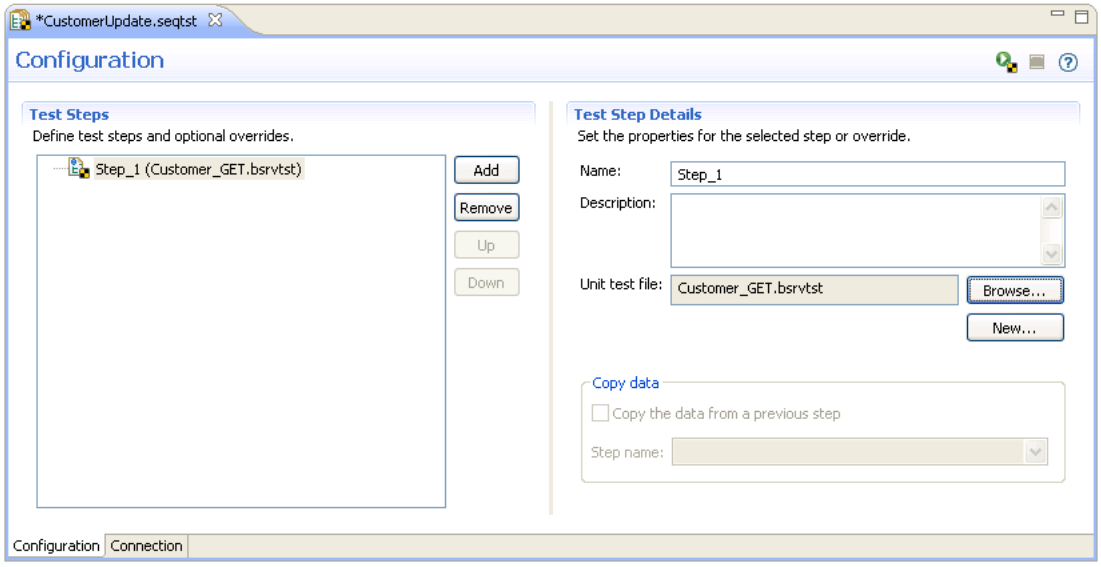
1 Select **Add**.

The **Test Step Details** section is displayed. For example:




2 Select **Browse** for **Unit test file**.

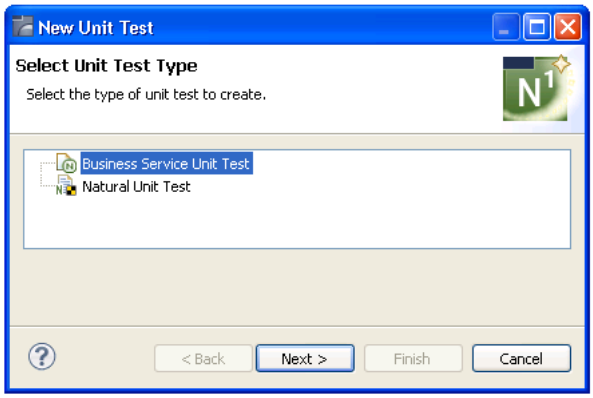
The **Select Unit Test** window is displayed. Select the unit test file and **OK**. The unit test details are displayed in the **Test Steps** section and the selected unit test file is displayed in **Unit test file**. For example:




Or:

Select **New** for **Unit test file**.

The **Select Unit Test Type** panel is displayed. For example:




Select one of the following options:

- **Business Service Unit Test**

  The **Define Business Service Unit Test Details** panel is displayed. For information, see *Create a Unit Test for a Business Service*.

- **Natural Unit Test**

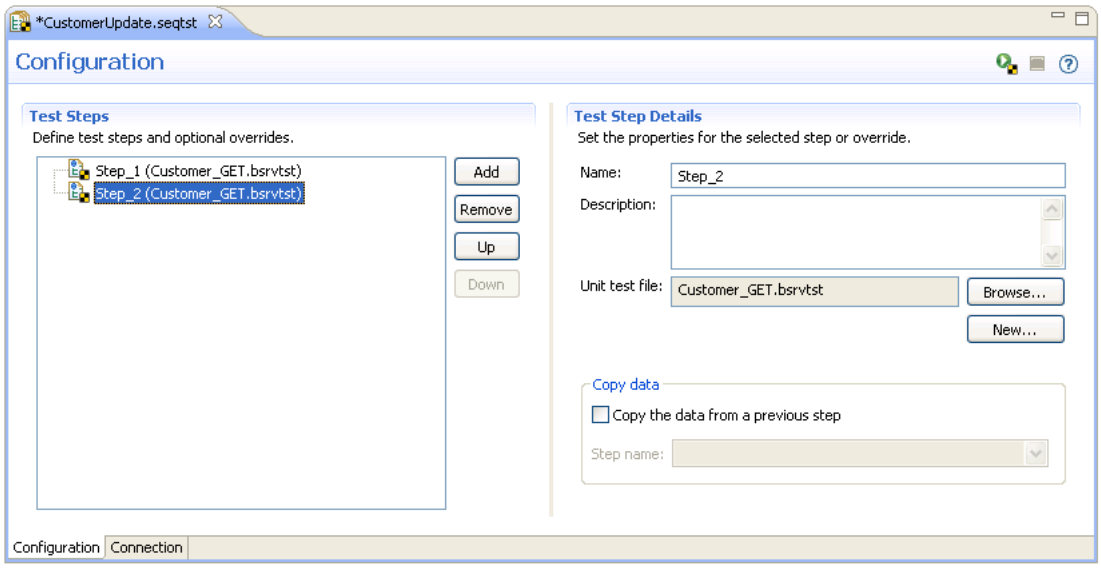
  The **Define Natural Unit Test Details** panel is displayed. For information, see *Create a Unit Test for a Subprogram*.

**Note:** When accessing these panels from the sequence unit test editor, the project name defaults to the name of the project containing the sequence unit test and is read-only. The unit test file specified for each test step must contain a relative path to the Testing-Suites folder in the project containing the sequence unit test.

After defining the unit test and selecting **Finish**, the unit test details are displayed in the **Test Steps** section and the newly created unit test file is displayed in **Unit test file**.

3 Select **Add**.

The second test step is displayed in **Test Steps** and the **Copy data** section is enabled. For example:




4 Select or create the unit test for the second test step.

Repeat steps 1 and 2 until all test steps have been added. Optionally, you can use this editor to:

| Task | Procedure |
|---|---|
| Provide a description of this test step. | Type a description of the test step in **Description** (maximum of 250 characters). The first 60 characters are displayed as the tool tip for the test step in **Test Steps**. |
| Copy data from a previous step. | See *Copy Data from a Previous Step*. |
| Delete a test step. | Select the test step in **Test Steps** and select **Remove** or open the context menu for the test step and select **Delete**. |
| Reorder the test steps. | Select the test step in **Test Steps** and select **Up** or **Down**. |
| Provide a name for the test step. | Type the step name in **Name**. |
| Define an input override for a field used in a test step. | See *Add an Input Override*. |
| Define a validation override for a field used in a test step. | See *Add a Validation Override*. |
| Define a method override for a method used in a test step (business service unit tests only). | See *Add a Method Override*. |

5 Save the settings.

### Copy Data from a Previous Step

This section describes how to copy data from a previous test step. When the generated sequence test is run, the test step will attempt to copy the data from the specified test step. If the test steps share the same Natural unit test file, the entire data structure from the previous test step is copied. If the test steps use different Natural unit test files, each field is copied by name and the level 1 name (if present) is compared to the field name.

**Caution:** All values are copied, even when the Natural formats are different. This may result in conversion errors (for example, when alpha values are placed in numeric fields).

#### To copy data from a previous test step

1 Select the test step to which you want to copy the data.

2 Select **Copy data from a previous step**.

3 Select the test step from which you want to copy the data in **Step name**.

You can select any previous test step in the list. Only previous test steps are listed, as data cannot be copied from a test step that has not been run.

**Note:** When defining input or validation overrides, you can also select the field from which to copy the data.

### Add an Input Override

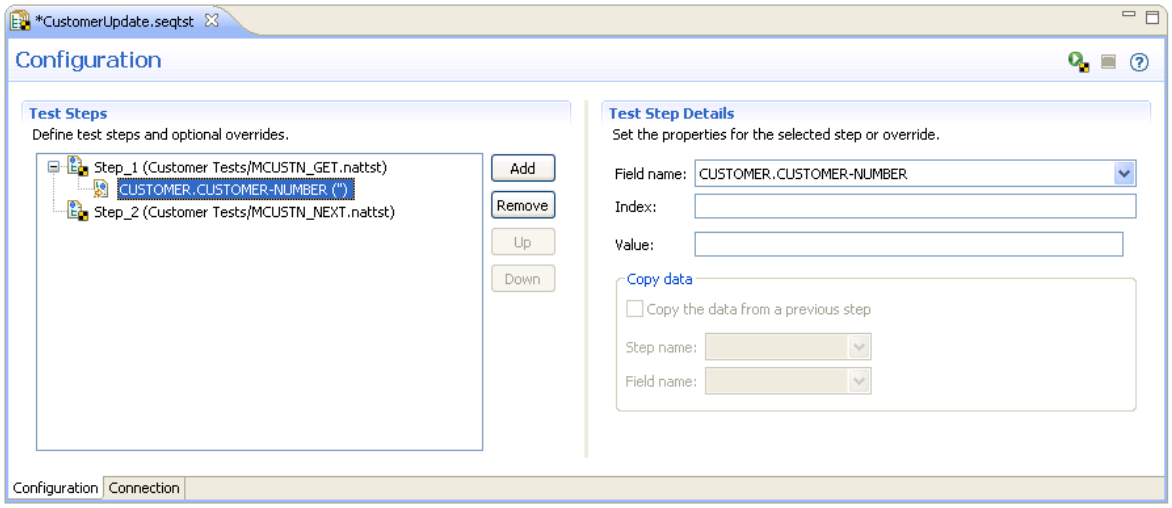
This section describes how to add an input override for a field. This value will override any input value defined for an input field with the same name in the original unit test file. For example, if the original unit test file has an input field and value of `FUNCTION=GET` and you add an override to a test step that sets `FUNCTION=UPDATE`, then `FUNCTION=UPDATE` will be used.

#### To add an input override

1 Open the context menu for the test step in **Test Steps**.

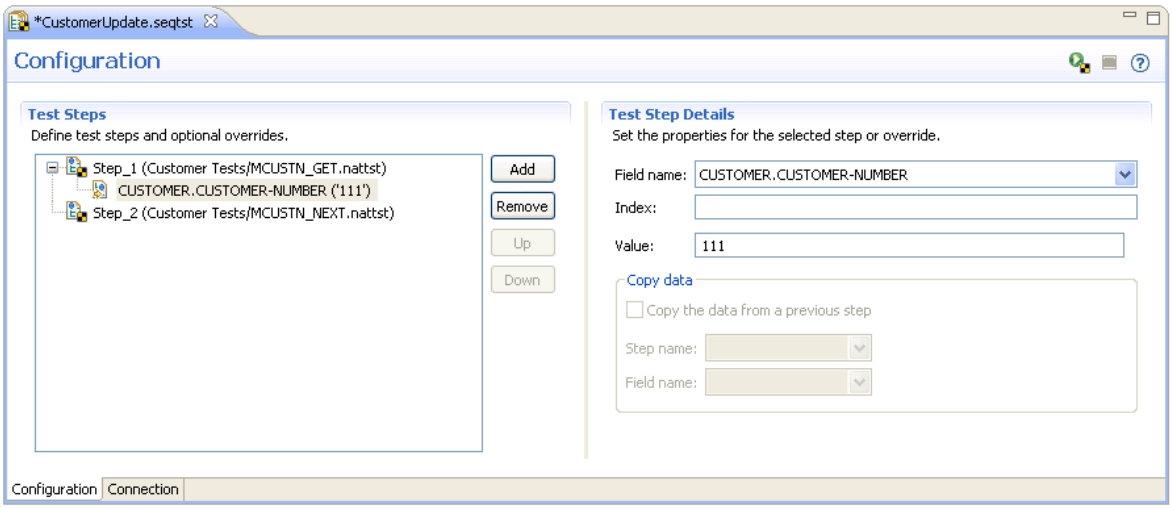
2 Select **New > Input Override**.

The field details are displayed in **Test Step Details**. For example:




3 Type the override value in **Value**.

The input override is displayed in **Test Steps**. For example:




In this example, an override value for the CUSTOMER-NUMBER field has been added.

**Notes:**

1. For information about the input parameters, see *Configure Input Parameters*.
2. You can copy the field data from a previous step. For information, see *Copy Data from a Previous Step*.
3. To remove an input override, either select the override in **Test Steps** and select **Remove** or open the context menu for the override and select **Delete**.

## Add a Validation Override

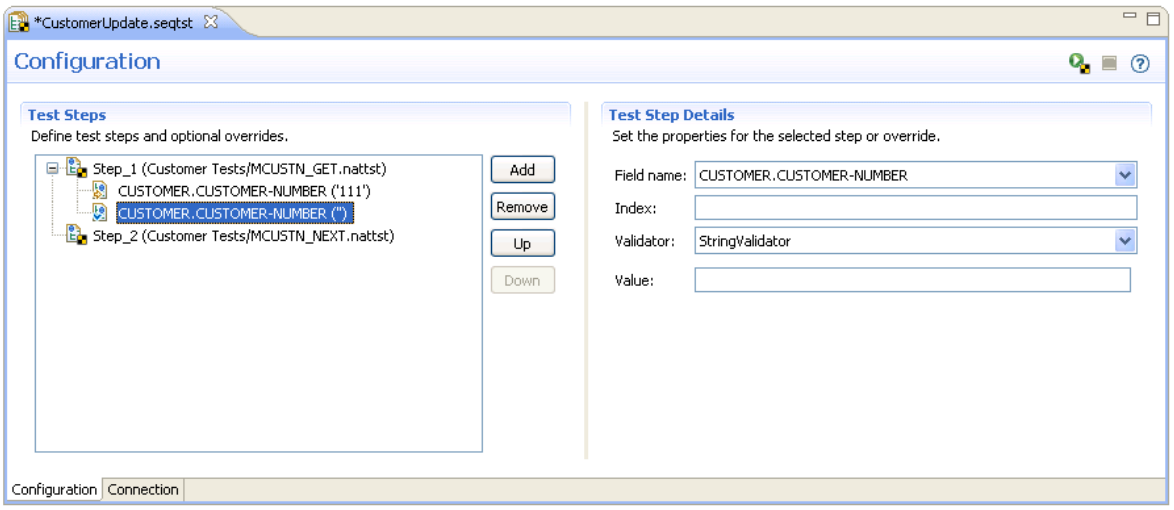
This section describes how to add an override value for a field validation. This value will override any validation defined for an input field with the same name in the original unit test file. For example, if the original unit test file has a field validation of `#MSG <> ERROR` and you add a validation override of `#MSG <> WARNING`, then both validations will be used (i.e., the wizard will ensure that the message is not equal to both ERROR and WARNING).

### To add a validation override

1 Open the context menu for the test step in **Test Steps**.

2 Select **New > Validation Override**.

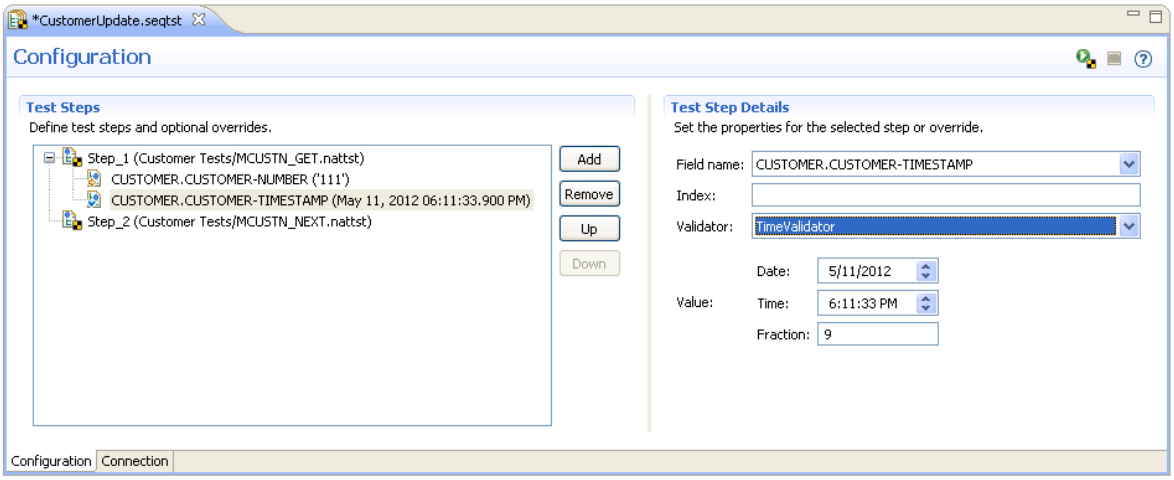
The validation details are displayed in **Test Step Details**. For example:




3 Select the field name in **Field name**.

4 Select the override value in **Validator**.

The validation override is displayed in **Test Steps**. For example:




In this example, an override validation for the CUSTOMER-TIMESTAMP field has been added.

**Notes:**

1. For information about the validation parameters, see *Define Validations*.
2. You can copy the validation data from a previous step. For information, see *Copy Data from a Previous Step*.
3. To remove a validation override, either select the override and select **Remove** or open the context menu for the override and select **Delete**.

## Add a Method Override

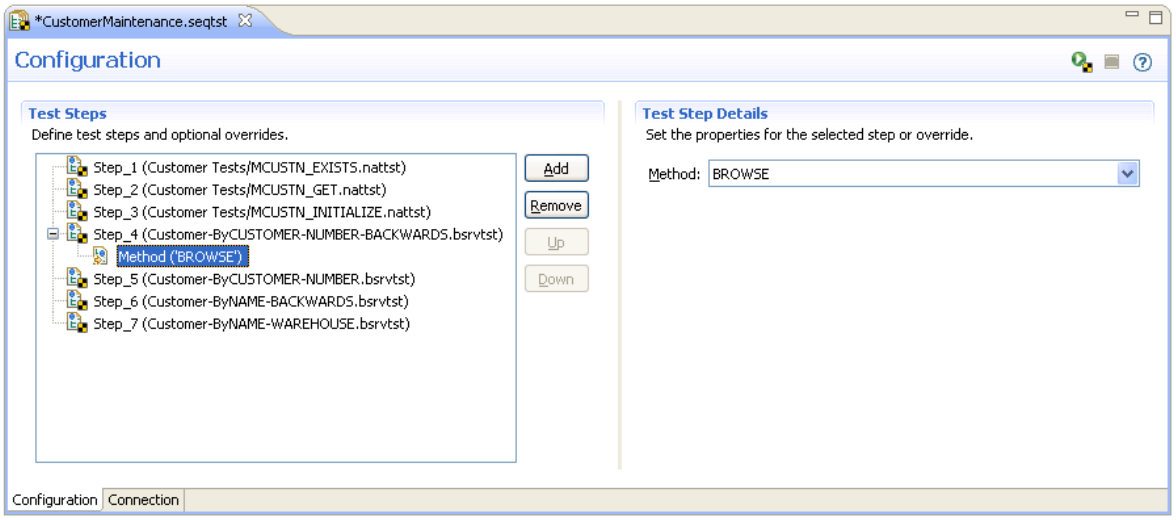
This section describes how to add a method override value for a business service unit test. This value will override the method name in the original business service unit test. For example, if the original unit test has a method value of "BROWSE" and you add a method override value "EXISTS" to a test step, then the sequence unit test will execute the "EXISTS" method.

### To add a method override

1 Open the context menu for the test step in **Test Steps**.

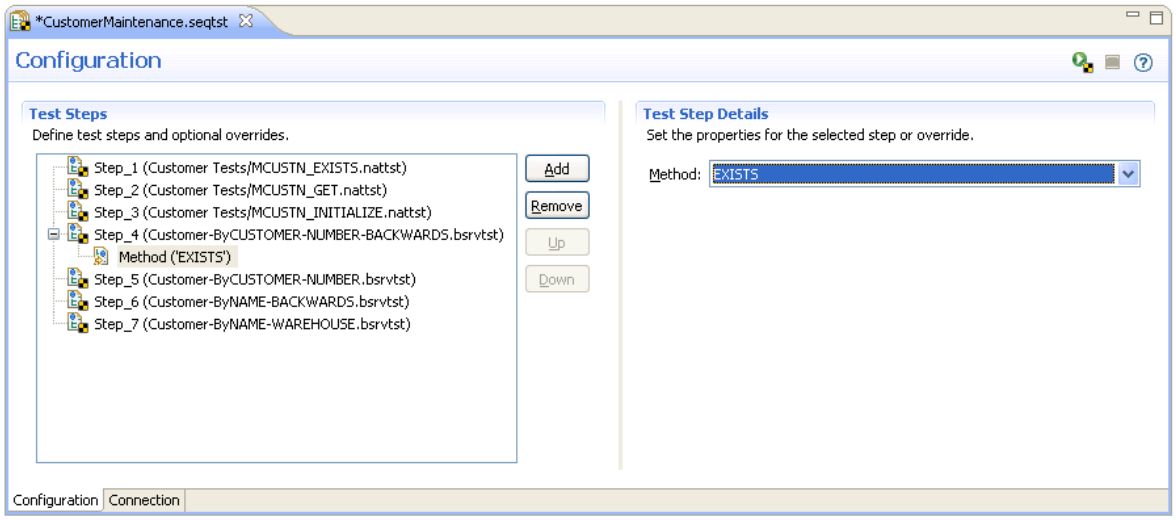
2 Select **New > Method Override**.

The method details are displayed in **Test Step Details**. For example:




3 Type the override value in **Method**.

The method override is displayed in **Test Steps**. For example:




In this example, an override value of METHOD=EXISTS has been added.

**Notes:**

1. For information about business service methods, see NaturalONE's *Business Services* documentation.
2. To remove a method override, either select the override in **Test Steps** and select **Remove** or open the context menu for the override and select **Delete**.

## Use the Dependencies View

When a generated module is open in the editor, the **Dependencies** view displays dependencies between a sequence unit test and the unit tests executed for each test step. This section describes the nodes contributed to the view for these resources. The following topics are covered:

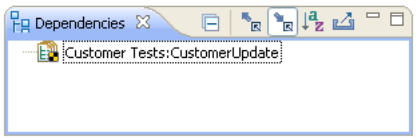
- Sequence Unit Test Resources
- Business Service Unit Test Resources
- Natural Unit Test Resources

**Notes:**

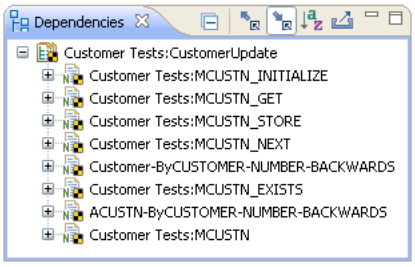
1. Select to sort the resources alphabetically.
2. Select to export a textual representation of the visible nodes in the view to a file.
3. When a supporting resource cannot be found locally using the project steplib chain and project references, "<Unknown>" is displayed with the name of the resource. If the unknown module(s) is not shipped with the Construct runtime project, either manually download it from the server or create it locally. If the module(s) is shipped with the Construct runtime project, add the project. For information, see NaturalONE's *Code Generation* documentation.
4. For more information about the **Dependencies** view, see the description of the source editor in *Using NaturalONE*.

### Sequence Unit Test Resources

When a sequence unit test is open in the editor view, the root node displays the name of the sequence unit test. In caller mode ( ), no child nodes are displayed because no other **Dependencies** view objects depend on this sequence unit test file. For example:

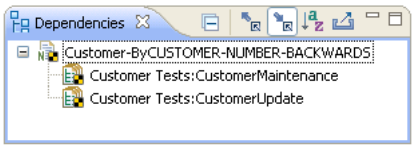



In callee mode ( ), the child nodes display one business service or Natural unit test for each test step in the sequence unit test. For example:

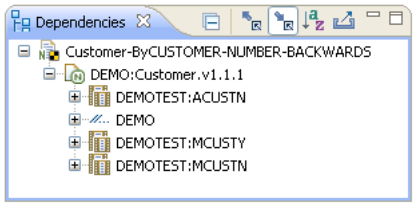



### Business Service Unit Test Resources

When a business service unit test is open in the editor view, the root node displays the name of the unit test. In caller mode (), one child node is displayed for each sequence unit test that includes this unit test in one of its test steps. For example:

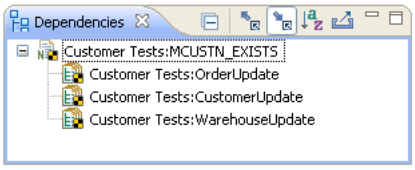



In callee mode (), the child node displays the name of the business service that the unit test executes, along with the names of the supporting Natural resources and the names of the libraries and projects in which they are located. For example:

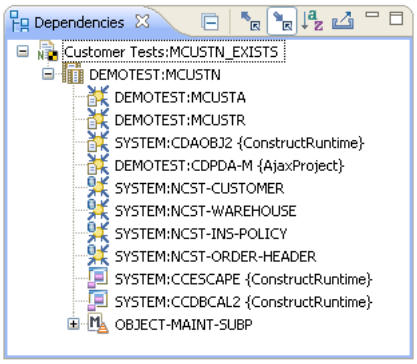



### Natural Unit Test Resources

When a Natural unit test is open in the editor view, the root node displays the name of the unit test. In caller mode (), one child node is displayed for each sequence unit test that includes this unit test in one of its test steps. For example:




In callee mode ( ), the child node displays the name of the Natural subprogram that the unit test executes, along with the names of the supporting Natural resources and the names of the libraries and projects in which they are located. For example:



# 9 Test an External Subroutine


- Access the Subroutine Tester ..... 114
- Test with a Program ..... 115
- Test with a Subprogram ..... 116

This section describes how to test an external subroutine. The tester can test the subroutine using either a subprogram or a program that calls a subprogram. The following tables describes which option to use:

| External Subroutine Features | Test Using |
|---|---|
| No parameters and screen input/output | Program (Natural for Ajax provides the screen input/output) |
| Parameters and no screen input/output | Subprogram (then you can use the subprogram tester to create scripts so the tests can be run again)<br>**Note:** If there are parameters and no screen input/output, it is easier to test the routine as a subprogram because the subprogram tester can handle the variety of parameters. |

Regardless of which option you use, temporary Natural objects are created to perform the tests and then deleted when the Natural for Ajax page or subprogram tester is closed.

**Note:** If you intend to use the temporary subprograms to create a unit (batch) test for the subroutine, save the files locally before closing the tester.

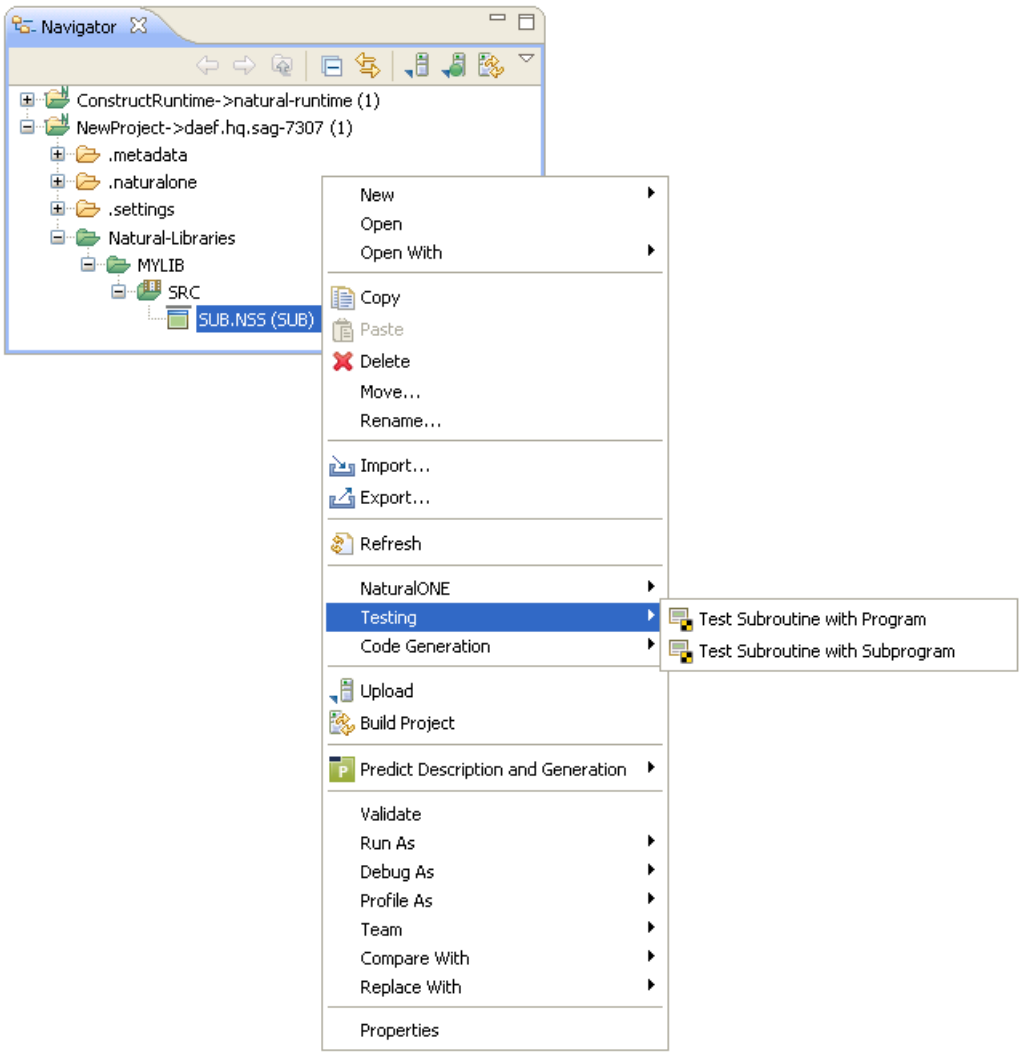
## Access the Subroutine Tester

### To access the subroutine tester

1 Open the context menu for the subroutine in the **Navigator** view.

2 Select **Testing**.

The test options for external subroutines are displayed. For example:



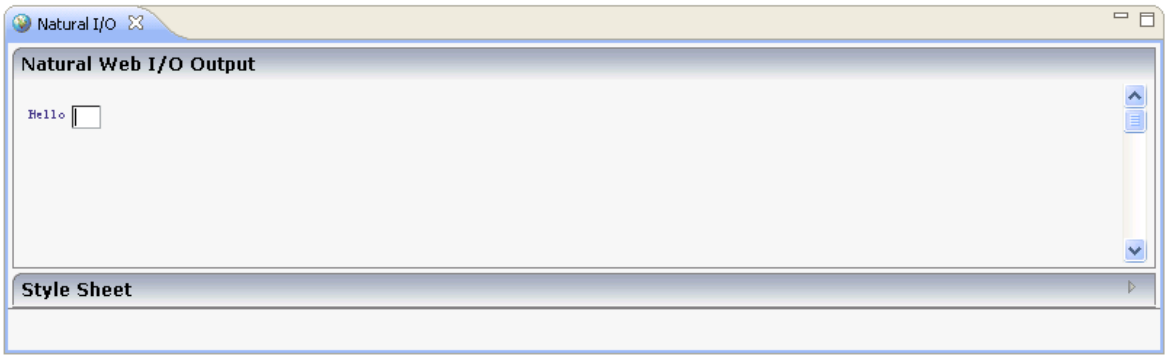

## Test with a Program

### To test an external subroutine using a program

1 Open the context menu for the subroutine in the **Navigator** view.

2 Select **Testing > Test Subroutine with Program**.

The subroutine is tested and the results are displayed in the **Natural I/O** view. For example:



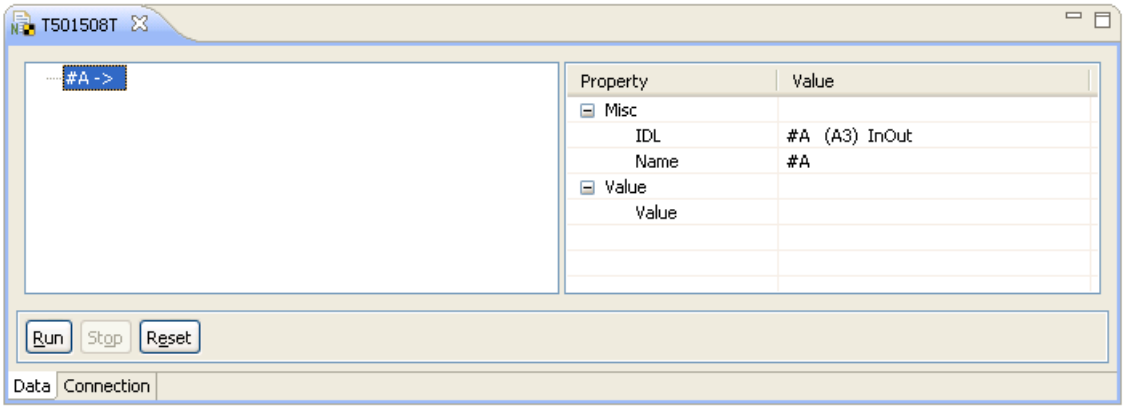

## Test with a Subprogram

### To test an external subroutine using a subprogram

1 Open the context menu for the subroutine in the **Navigator** view.

2 Select **Testing > Test Subroutine with Subprogram**.

The tester creates a temporary subprogram file to test the subroutine. For example:




**Note:** This editor functions in the same way as the editor used to test a subprogram. For information on using this editor, see *Features of the Test Editors* and *Test a Subprogram Directly*.

# 10 Test a Natural Map

This section describes how to test a Natural map in NaturalONE. The tester allows you to test a map as you would on the server (i.e., pressing PF4 in the map editor).

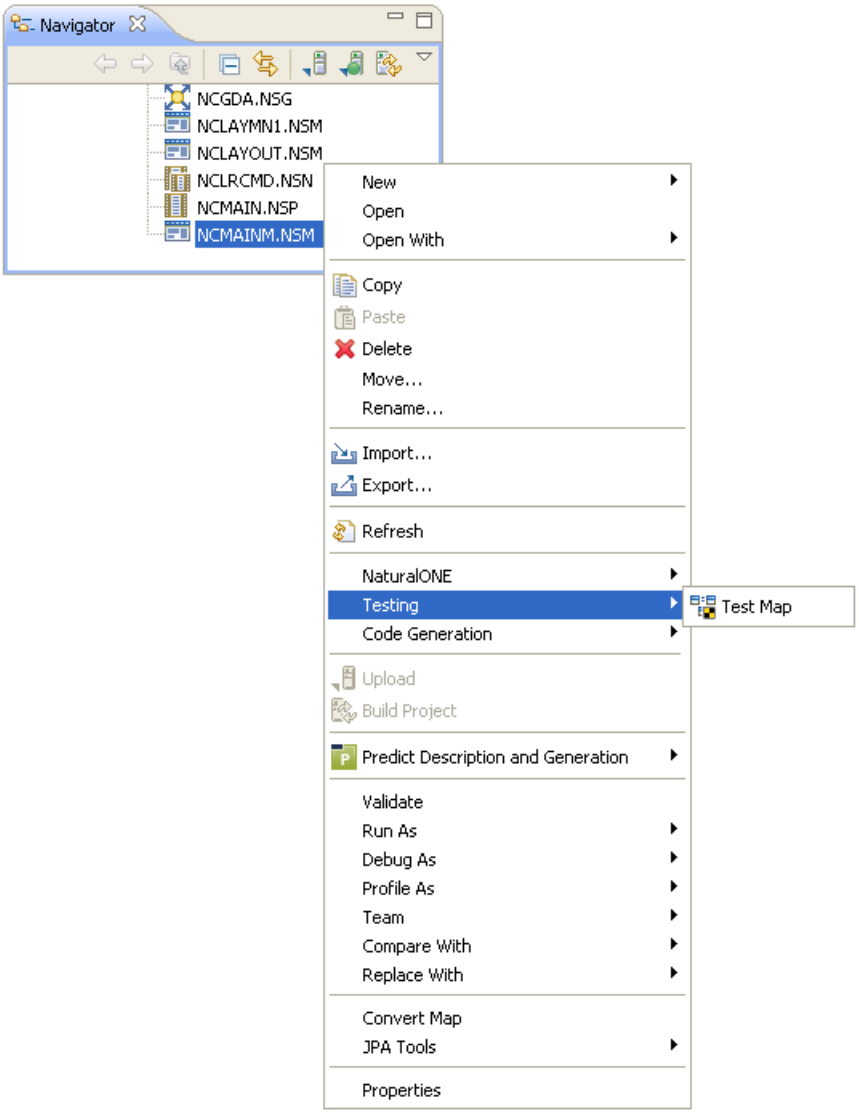
**Note:** The map must be available locally. If the map is not available locally, download it from the server.

### To test a Natural map

1 Open the context menu for the map in the **Navigator** view.

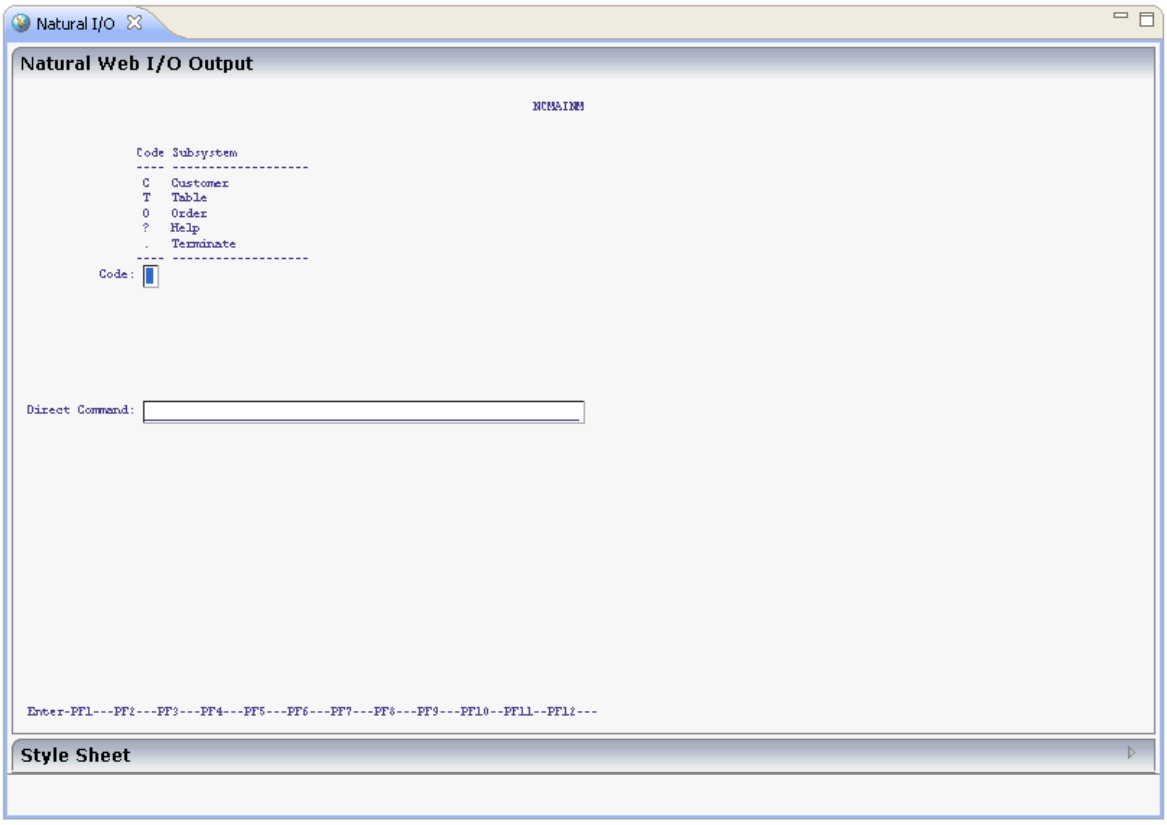
2 Select **Testing**.

For example:



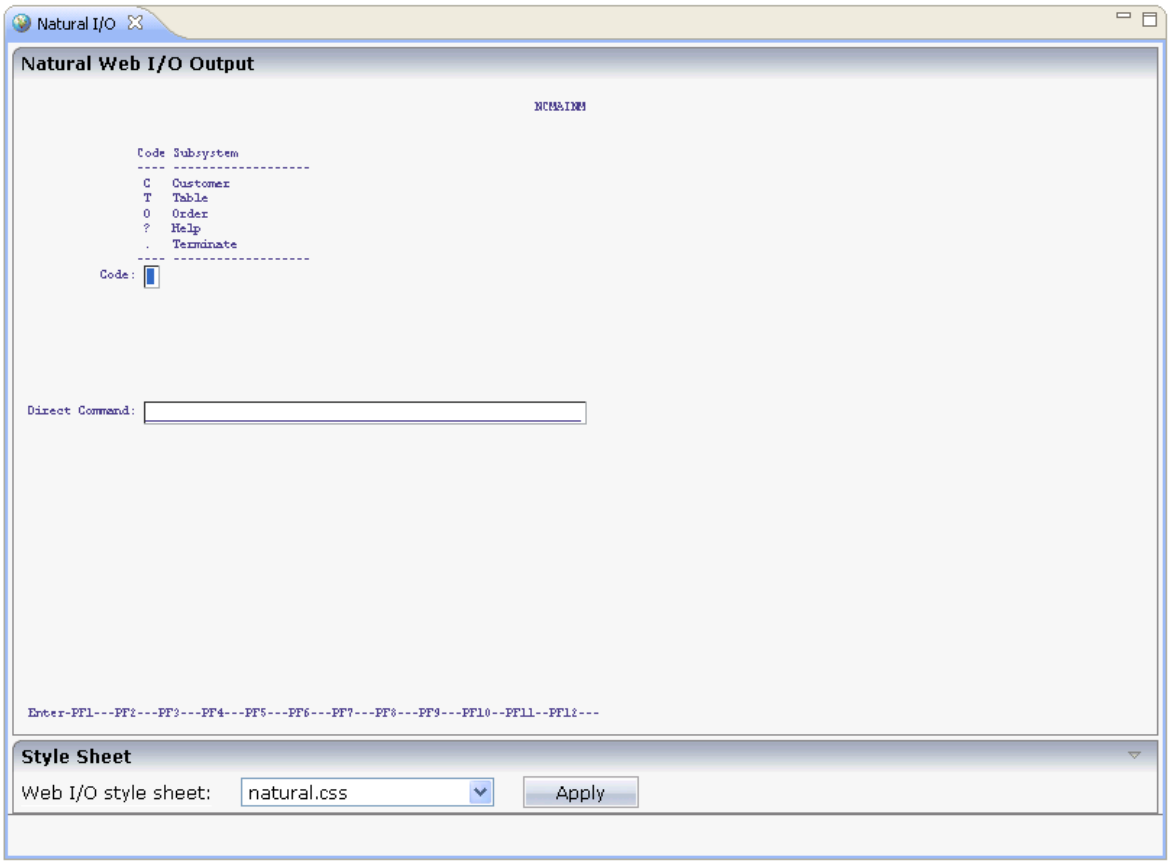

3 Select **Test Map**.

The output of the map is displayed. For example:




In addition to testing the output of the map, you can also test all code within the map. For example, you can enter "?" in an input field to display the available help information (if help has been attached to the map).

You can also apply a different style sheet to the map by selecting ▸ in **Style Sheet**. For example:




In this example, the default style sheet *natural.css* has been used. If you would like to see the same colors in the output window as in the map editor, you can use the style sheet *natural_mapeditor.css* instead of the default style sheet.

To change style sheets, select the file in **Web I/O style sheet** and select **Apply**. The map is redisplayed with the selected style sheet.

# 11 Setting Preferences for Application Testing


- Showing the Preferences for Application Testing ..... 122
- Set Logging Preferences for Unit Tests ..... 123
- Set Server Synchronization Preferences ..... 123

This section describes how to set preferences for the supplied test function.

## Showing the Preferences for Application Testing

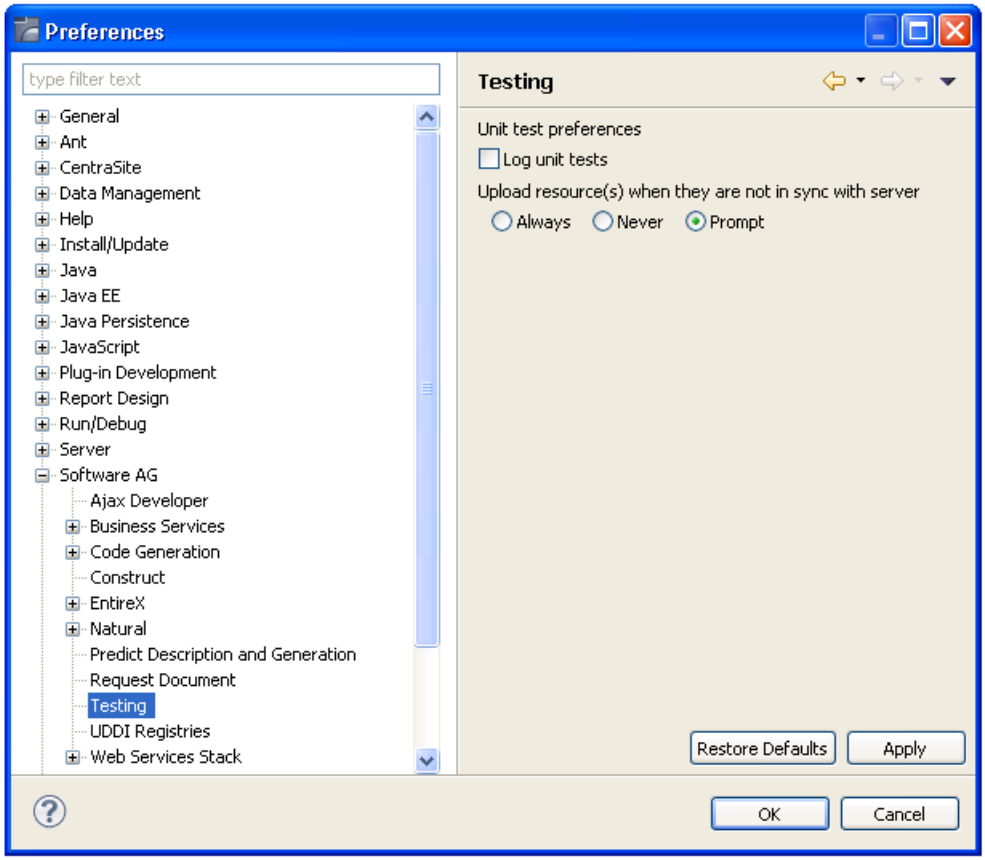
The preferences for Application Testing are set in the **Preferences** dialog box of Eclipse.

### To show the preferences for Application Testing

1 From the **Window** menu, choose **Preferences**.

2 In the tree of the resulting dialog box, expand the **Software AG** node and then select the **Testing** node.

The **Testing** page is displayed.



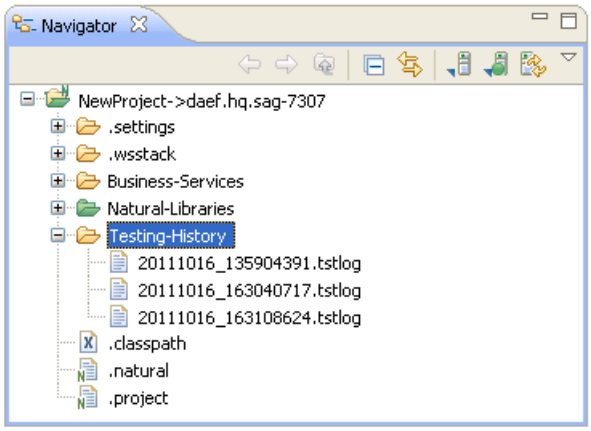
## Set Logging Preferences for Unit Tests

### To set logging preferences

1 Display the **Testing** page as described above.

2 Select **Log unit tests**.

Unit test log files will be created automatically each time a unit test is executed. The log files are stored in the **Testing-History** folder within the NaturalONE project in which the unit test was executed and include a *.tstlog* file extension. For example:




**Note:** If this option is not selected, the log files will not be created.

3 Select **OK** to save the preferences.

## Set Server Synchronization Preferences

When testing a subprogram, a message may be displayed indicating that a local resource has not been uploaded to the server and synchronized with the server resource. You can set preferences for this option.

### To set server synchronization preferences

1 Display the **Testing** page as described above.

2 Select one of the options listed in **Upload resource(s) when they are not in sync with server**.

These options are:

| Option | Description |
|---|---|
| Always | Resource(s) are always uploaded to the server when not in sync. |
| Never | Resource(s) are never uploaded to the server when not in sync. |
| Prompt | A window is displayed to select an option. |

3 Select **OK** to save the preferences.

# 12 Creating Ant Scripts to Run Unit Tests


- Set Up the Environment ..... 126
- Generate the Ant Script and Command Files ..... 126
- Define the testsuite Ant Task ..... 130

You can use the Ant script wizard to create XML-based Ant scripts to run unit test files (file extension *.bsrvtst*, *.exttst*, *.nattst*, and *.seqtst*). You can run the Ant script from within NaturalONE or use the external command file to execute tests via a command line. The wizard generates the following files:

- *build.xml* (contains the Ant script)
- *run.cmd* (contains the DOS command file to run the script)

For information on creating unit test files, see:

- ***Create a Unit Test for a Business Service***
- ***Create a Unit Test for a Subprogram***
- ***Create an External Data Unit Test***
- ***Create a Sequence Unit Test***

## Set Up the Environment

If you use an RPC environment connection ID, the ID must be setup before running the wizard. For information, refer to the EntireX documentation.

## Generate the Ant Script and Command Files

This section describes how to create the *build.xml* and *run.cmd* files.

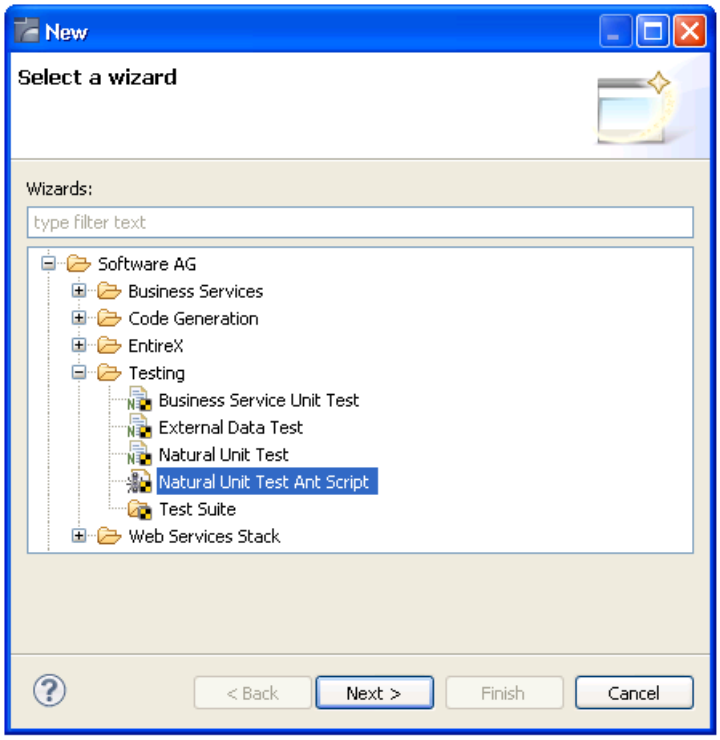
### To create the Ant script

1 Select **New > Other** on the **File** menu.

The **Select a wizard** panel is displayed.

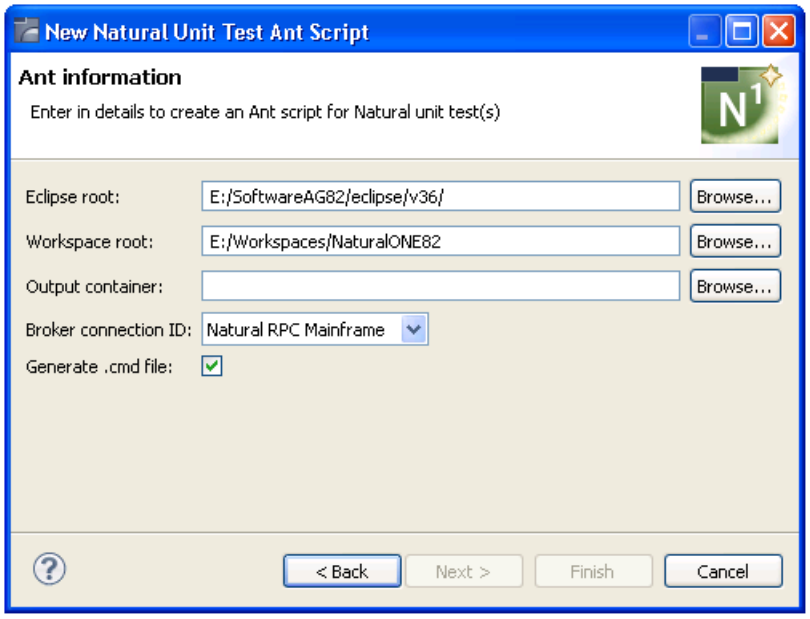
2 Select **Software AG > Testing > Natural Unit Test Ant Script**.

For example:




3 Select **Next**.

The **Ant information** panel is displayed. For example:



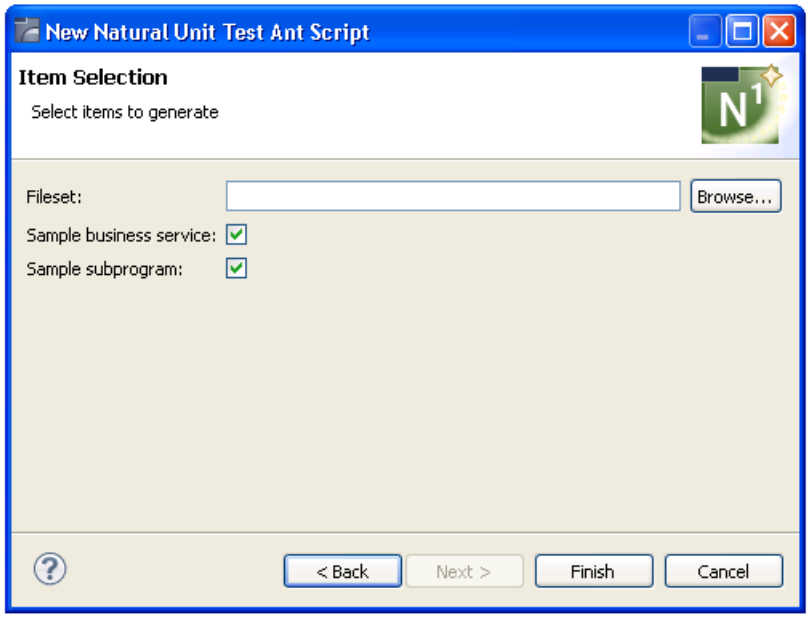

Using this panel, you can:

| Task | Procedure |
|---|---|
| Change the location of the default root folder in Eclipse. | Select a new folder in **Eclipse root**. |
| Change the location of the default workspace folder. | Select a new folder in **Workspace root**. |
| Change the Broker connection ID. | Select a new ID in **Broker connection ID**. |
| Suppress the generation of the *run.cmd* file containing the DOS command file that runs the script. | Deselect **Generate .cmd file**. |

4 Type the name of the Natural project in **Output container** or select **Browse** to display a list of available projects for selection.

5 Select **Next**.

The **Item Selection** panel is displayed. For example:



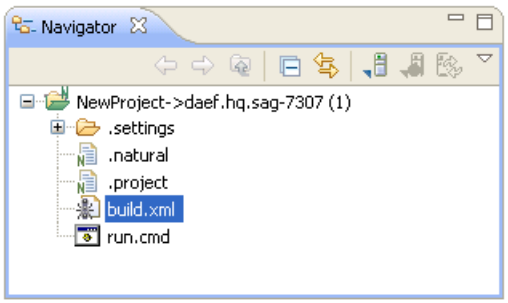

Using this panel, you can:

| Task | Procedure |
|---|---|
| Suppress the generation of a sample business service. | Deselect **Sample business service**. |
| Suppress the generation of a sample subprogram. | Deselect **Sample subprogram**. |

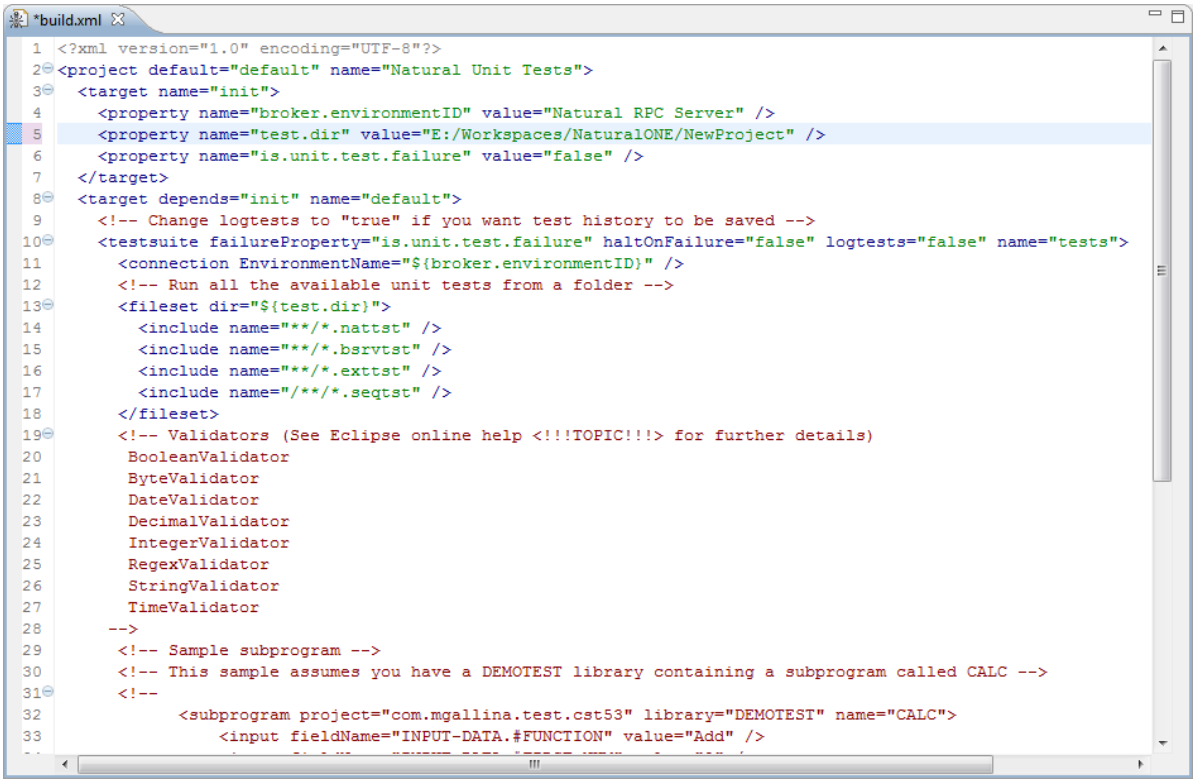
6 Select the location of the default fileset folder (the workspace root folder and the output container folder) in **Fileset**.

7 Select **Finish**.

The generated *build.xml* and *run.cmd* files are displayed in the **Navigator** view. For example:




The *build.xml* file is displayed in the editor view. For example:




8 Refine the parameters for the testsuite Ant task as desired.

# Define the testsuite Ant Task

This section describes the parameters for the testsuite Ant task in the generated *build.xml* file. The following topics are covered:


- Description
- Parameters
- Parameters Specified as Nested Elements


## Description

Represents the set of Natural unit tests to be run.

It is assumed that all necessary resources to run the tests are contained within a NaturalONE project. To run subprogram tests, a local copy of the subprogram file (*.nsn file*) must be in the correct Natural Library folder. To run business service tests, the folder must contain the domain file, steplib file associated with the domain, and all subprogram file(s) referenced by the business service.

Each testsuite contains a connection node that defines how the tests will connect the Natural server.

There are three ways to run Natural unit tests:

- Create the units tests in NaturalONE using one of the Unit test wizards and then add a fileset subnode that will load the generated *.bsrvtst*, *.exttst*, or *.nattst* files
- Add a subprogram node to test a specific subprogram
- Add a businessService node to test a specific business service

## Parameters

| Attribute | Description | Required |
|---|---|---|
| failureProperty | Name of the Ant property that will be set to "true" when one or more unit tests fail. | No |
| haltOnFailure | Value indicating whether to halt the Ant script by throwing an Ant build exception upon receiving the first unit test failure. Valid values are "true" (halt the Ant script upon first failure) or "false" (run all unit tests regardless of failure). By default, this option is false. | No |
| logtests | Value indicating whether to log and save test history to the **Testing-History** folder. Valid values are "true" (save test history) or "false" (do not save test history). By default, this option is false. | No |
| name | Name used by the testsuite for output information in the test logs and Ant build log. | No |

| Attribute | Description | Required |
|---|---|---|
| setProjectSteplibs | Value indicating whether the steplibs from the Natural project are set in the RPC server environment.<br><br>**Note:** Keep in mind that the Natural Development Server used by the project must always be accessible when the value is `ON` (or when this property is not specified at all). The Natural Development Server is accessed to check the development mode settings for the steplib consolidation. These steplibs are then passed to the RPC server. In order to be compatible with older *build.xml* files, the default value is `ON` when not specified. | No |

### Parameters Specified as Nested Elements

This section describes parameters that are specified as nested elements. The following topics are covered:

- connection
- fileset
- subprogram
- businessService
- input
- validate

#### connection

Defines the connection settings to use to communicate with the Natural server.

Parameters:

| Attribute | Description | Required |
|---|---|---|
| address | Broker address (when a broker ID is specified). | Mandatory when brokerID is used |
| brokerID | Broker ID for the connection. | Either brokerID or environmentName |
| environmentName | Name of an EntireX RPC connection configured in Eclipse. | Either environmentName or brokerID |
| logon | Whether a Natural logon is required. | Optional and only when brokerID is used |
| password | Password for the connection. | No |
| rpcUserID | RPC user ID for the connection. | Optional (used with secured Natural environments) |
| rpcPassword | RPC password for the connection. | Optional (used with secured Natural environments) |
| userID | User ID for the connection. | Mandatory when brokerID is used |

#### fileset

Runs a set of unit test files.

Parameters:

| Attribute | Description | Required |
|---|---|---|
| dir | Name of the folder/project containing the unit test files. | Yes |

Parameters specified as nested elements:

| Parameter | Description |
|---|---|
| include name | Name of the unit test(s) to run. For example, `include name="**/*.bsrvtst" /` will run all business service unit tests in the specified folder/project. |

#### subprogram

Runs a single test against a subprogram.

Parameters:

| Attribute | Description | Required |
|---|---|---|
| project | Name of the Eclipse Natural project containing the subprogram. | Yes |
| library | Natural library containing the subprogram. | Yes |
| name | Name of the subprogram to execute, excluding the file extension (*.NSN*). | Yes |

Parameters specified as nested elements:

| Parameter | Description |
|---|---|
| input | See ***input***. |
| validate | See ***validate***. |

#### businessService

Runs a single test against a business service.

Parameters:

| Attribute | Description | Required |
|---|---|---|
| project | Name of the Eclipse Natural project containing the business service. | Yes |
| domain | Name of the domain containing the business service. | Yes |
| name | Name of the business service to run. | Yes |
| version | Version of the business service to run. | Yes |
| method | Name of the method to test. | Yes |

Parameters specified as nested elements:

| Parameter | Description |
|---|---|
| input | See ***input***. |
| validate | See ***validate***. |

**input**

Specifies the value for a field to be used for input.

Parameters:

| Attribute | Description | Required |
|---|---|---|
| name | Fully qualified field name in the format:<br>`[level one].[name]` | Yes |
| value | Value to assign to the field. | Yes |

**validate**

Specifies the field to be validated when it is returned by the call to the server.

Parameters:

| Attribute | Description | Required |
|---|---|---|
| name | Fully qualified field name in the format:<br>`[level one].[name]` | Yes |
| type | Type of validator to use (see the following table). | Yes |
| value | Value to assign to the field. | Yes |

Validators:

| Type | Description |
|---|---|
| BooleanValidator | Validates Boolean values. True values are: x, t , true, or 1. |
| ByteValidator | Compares an array of bytes. |
| ComparisonValidator | Compares values based on mathematical expressions (for example, ">", "<", "=", "<=", ">="). |
| DateValidator | Compares dates. The value is in the format: MON DD, YYYY (where MON is a 3-character abbreviation for a month name). |
| DecimalValidator | Compares decimal values. |
| IntegerValidator | Compares integer values. Decimals will be truncated. |
| RegexValidator | Verifies that the value in the field matches a regular expression. |
| StringValidator | Compares the value in the field against a string. |
| TimeValidator | Compares the value in the field against a time. Time is in the format: MMM d, yyyy hh:mm:ss.SSS. |